
devilbox Documentation

cytopia

Jan 25, 2019

| | | |
|----------|--------------------------------------|-----------|
| 1 | Read first | 3 |
| 1.1 | Shell commands | 3 |
| 1.2 | Checklists | 3 |
| 1.3 | Where to start? | 4 |
| 2 | Features | 5 |
| 2.1 | Projects | 6 |
| 2.2 | Service and version choice | 7 |
| 2.3 | Configuration | 7 |
| 2.4 | Intranet | 7 |
| 2.5 | Dockerized | 8 |
| 2.6 | Others | 8 |
| 3 | Install the Devilbox | 9 |
| 3.1 | Supported OS | 9 |
| 3.2 | Requirements | 10 |
| 3.3 | Download the devilbox | 10 |
| 3.4 | Create .env file | 11 |
| 3.5 | Adjust .env file | 11 |
| 3.6 | Checklist | 12 |
| 4 | Update the Devilbox | 13 |
| 4.1 | Update git repository | 14 |
| 4.2 | Update Docker images | 15 |
| 4.3 | Checklist git repository | 16 |
| 4.4 | Checklist Docker images | 16 |
| 5 | Start the Devilbox | 17 |
| 5.1 | Start all container | 17 |
| 5.2 | Start some container | 18 |
| 5.3 | Open Devilbox intranet | 18 |
| 5.4 | Checklist | 19 |
| 6 | Directory overview | 21 |
| 6.1 | Data directory | 21 |
| 6.2 | Project directory | 22 |
| 6.3 | Docroot directory | 22 |

| | | |
|-----------|---|-----------|
| 6.4 | Domain suffix | 22 |
| 6.5 | Making sense of it | 22 |
| 6.6 | Checklist | 23 |
| 7 | Create your first project | 25 |
| 7.1 | Step 1: visit Intranet vhost page | 26 |
| 7.2 | Step 2: create a project directory | 26 |
| 7.3 | Step 3: create a docroot directory | 27 |
| 7.4 | Step 4: create a DNS entry | 28 |
| 7.5 | Step 5: Visit your project | 29 |
| 7.6 | Step 6: Create a hello world | 30 |
| 7.7 | Checklist | 31 |
| 8 | Read log files | 33 |
| 8.1 | Mounted logs | 33 |
| 8.2 | Docker logs | 34 |
| 8.3 | Checklist | 34 |
| 9 | Email catch-all | 35 |
| 10 | Enter the PHP container | 37 |
| 10.1 | How to enter | 37 |
| 10.2 | How to become root | 38 |
| 10.3 | Tools | 38 |
| 10.4 | Advanced | 39 |
| 10.5 | Checklist | 39 |
| 11 | The Intranet | 41 |
| 11.1 | Devilbox tools | 42 |
| 11.2 | Third-party tools | 48 |
| 11.3 | Settings | 49 |
| 11.4 | Checklist | 49 |
| 12 | Best practice | 51 |
| 12.1 | Move data out of Devilbox directory | 51 |
| 12.2 | PHP project hostname settings | 54 |
| 12.3 | Timezone | 55 |
| 13 | Backup and restore MySQL | 57 |
| 13.1 | Backup | 58 |
| 13.2 | Restore | 61 |
| 14 | Backup and restore PostgreSQL | 63 |
| 14.1 | Backup | 63 |
| 14.2 | Restore | 64 |
| 15 | Backup and restore MongoDB | 67 |
| 15.1 | Backup | 67 |
| 15.2 | Restore | 68 |
| 16 | Communicating with external hosts | 69 |
| 16.1 | Prerequisites | 70 |
| 16.2 | Make DNS available to the Devilbox | 71 |
| 16.3 | Further reading | 72 |

| | |
|---|------------|
| 17 Add your own Docker image | 73 |
| 17.1 Prerequisites | 73 |
| 17.2 What information do you need? | 74 |
| 17.3 How to add a new service? | 74 |
| 17.4 How to start the new service? | 76 |
| 17.5 Further reading | 76 |
| 18 Overwrite existing Docker image | 77 |
| 18.1 Prerequisites | 77 |
| 18.2 What information do you need? | 77 |
| 18.3 How to overwrite a service? | 78 |
| 18.4 Further reading | 79 |
| 19 Adding Sub domains | 81 |
| 19.1 Single sub domain for one project | 82 |
| 19.2 Multiple sub domains for one project | 82 |
| 20 Change container versions | 91 |
| 20.1 Change PHP version | 91 |
| 20.2 Change whatever version | 93 |
| 20.3 Checklist | 93 |
| 21 Work inside the container | 95 |
| 21.1 Enter the container | 96 |
| 21.2 Inside the container | 96 |
| 21.3 Leave the container | 97 |
| 21.4 Host to Container mappings | 97 |
| 21.5 Checklist | 99 |
| 22 Enable Xdebug | 101 |
| 22.1 Enable Xdebug | 102 |
| 22.2 Configure your IDE | 105 |
| 23 Custom environment variables | 113 |
| 23.1 Add custom environment variables | 113 |
| 23.2 Use custom environment variables | 114 |
| 24 Static Code Analysis | 115 |
| 24.1 Awesome-ci | 115 |
| 24.2 PHPCS | 116 |
| 24.3 ESLint | 117 |
| 25 Setup CakePHP | 119 |
| 25.1 Overview | 119 |
| 25.2 Walk through | 120 |
| 26 Setup Drupal | 123 |
| 26.1 Overview | 123 |
| 26.2 Walk through | 124 |
| 27 Setup Joomla | 127 |
| 27.1 Overview | 127 |
| 27.2 Walk through | 128 |
| 28 Setup Laravel | 131 |
| 28.1 Overview | 131 |

| | | |
|-----------|--|------------|
| 28.2 | Walk through | 132 |
| 29 | Setup Phalcon | 135 |
| 29.1 | Overview | 135 |
| 29.2 | Walk through | 136 |
| 30 | Setup Symfony | 139 |
| 30.1 | Overview | 139 |
| 30.2 | Walk through | 140 |
| 31 | Setup Wordpress | 143 |
| 31.1 | Overview | 143 |
| 31.2 | Walk through | 144 |
| 32 | Setup Yii | 147 |
| 32.1 | Overview | 147 |
| 32.2 | Walk through | 148 |
| 33 | Setup Zend | 151 |
| 33.1 | Overview | 151 |
| 33.2 | Walk through | 152 |
| 34 | DNS records | 155 |
| 34.1 | Examples | 156 |
| 34.2 | Creating DNS records | 156 |
| 34.3 | Verify | 158 |
| 35 | Customized virtual host (vhost-gen) | 159 |
| 35.1 | vhost-gen | 160 |
| 35.2 | Templates explained | 162 |
| 35.3 | Apply Changes | 165 |
| 35.4 | Further readings | 166 |
| 36 | HTTPS (SSL) | 167 |
| 36.1 | TL;DR | 167 |
| 36.2 | How does it work | 168 |
| 36.3 | Import the CA into your browser | 168 |
| 36.4 | Further Reading | 175 |
| 37 | Web server | 177 |
| 37.1 | Features | 178 |
| 37.2 | Information | 178 |
| 37.3 | Configuration | 178 |
| 38 | PHP | 179 |
| 39 | MySQL | 181 |
| 40 | MongoDB | 183 |
| 41 | Redis | 185 |
| 42 | Memcached | 187 |
| 43 | BIND | 189 |
| 44 | Devilbox Intranet | 191 |

| | |
|---|------------|
| 45 Auto-DNS | 193 |
| 45.1 Native Docker | 193 |
| 45.2 Docker Toolbox | 196 |
| 46 .env file | 199 |
| 46.1 Core settings | 201 |
| 46.2 Intranet settings | 206 |
| 46.3 Docker image versions | 208 |
| 46.4 Docker host mounts | 212 |
| 46.5 Docker host ports | 215 |
| 46.6 Container settings | 217 |
| 47 docker-compose.yml | 225 |
| 48 docker-compose.override.yml | 227 |
| 48.1 Create docker-compose.override.yml | 227 |
| 48.2 Further reading | 228 |
| 49 apache.conf | 229 |
| 49.1 General | 229 |
| 49.2 Examples | 230 |
| 50 nginx.conf | 233 |
| 50.1 General | 233 |
| 50.2 Examples | 234 |
| 51 php.ini | 237 |
| 51.1 General | 237 |
| 51.2 Examples | 238 |
| 52 php-fpm.conf | 239 |
| 52.1 General | 239 |
| 52.2 Examples | 240 |
| 53 my.cnf | 243 |
| 53.1 General | 243 |
| 53.2 Examples | 244 |
| 54 bashrc.sh | 247 |
| 54.1 Directory mapping | 247 |
| 54.2 Examples | 248 |
| 55 Docker and Docker Compose | 251 |
| 55.1 Install Docker | 251 |
| 55.2 Install Docker Compose | 252 |
| 55.3 Checklist | 252 |
| 56 Docker Toolbox | 253 |
| 56.1 Installation | 253 |
| 56.2 Additional steps | 254 |
| 56.3 Checklist | 254 |
| 57 Available container | 257 |
| 58 Available tools | 259 |

| | |
|--|------------|
| 59 Remove stopped container | 261 |
| 59.1 Why should I? | 261 |
| 59.2 How to do it? | 261 |
| 59.3 When to do it? | 261 |
| 60 Synchronize container permissions | 263 |
| 60.1 Unsynchronized permissions | 263 |
| 60.2 It gets even worse | 264 |
| 60.3 The solution | 264 |
| 61 FAQ | 265 |
| 61.1 General | 266 |
| 61.2 Configuration | 267 |
| 61.3 Compatibility | 268 |
| 62 Troubleshooting | 271 |
| 62.1 Invalid bind mount spec | 271 |
| 62.2 [Warning] World-writable config file ‘etc/mysql/docker-default.d/my.cnf’ is ignored | 272 |
| 63 Blogs, Videos and Use-cases | 273 |
| 63.1 Official videos | 273 |
| 63.2 Blog posts | 273 |
| 63.3 Use-cases | 274 |
| 63.4 Add your story | 274 |
| 64 Artwork | 275 |



The Devilbox is a modern dockerized LAMP and MEAN stack for local development on Linux, MacOS and Windows.

It allows you to have an unlimited number of projects ready without having to install any external software and without having to configure any virtual hosts. As well as providing a very flexible development stack that you can run offline. (Internet is only required to initially pull docker container).

The only thing you will have to do is to create a new directory on the filesystem and your virtual host is ready to be served with your custom domain.

Important:

Read first Ensure you have read this document to understand how this documentation works.

Find some useful information and tips for the documentation itself.

1.1 Shell commands

Important: All shell commands in this documentation use two different formats:

1. This one indicates that the command should be executed on your host operating system. (When copying commands, do not copy the `host>` part).

```
host> command
```

2. This one indicates that the command should be executed inside the currently selected PHP container. (When copying commands, do not copy the `php>` part).

```
php> command
```

1.2 Checklists

Note: Most guides and tutorials provide a **Checklist** at the very bottom. You can as well jump to it and quickly see if you have done everything already.

1.3 Where to start?

On the left menu you will find a `GETTING STARTED` section, **read through all of them** to get a basic theoretical and practical understanding about the Devilbox.

There is also a *Blogs, Videos and Use-cases* section that might be useful as an additional crash-course.

This section gives you a brief overview about the available features.

Table of Contents

- *Projects*
 - *Unlimited projects*
 - *Automated virtual hosts*
 - *Automated DNS records*
 - *Email catch-all*
 - *Log files*
 - *Virtual host domains*
- *Service and version choice*
 - *Selective start*
 - *Version choice*
 - *LAMP and MEAN stack*
- *Configuration*
 - *Global configuration*
 - *Version specific configuration*
 - *Project specific configuration*
- *Intranet*
 - *Command & Control Center*
 - *Third-party tools*

- *Dockerized*
 - *Portable*
 - *Built nightly*
 - *Ships popular development tools*
 - *Work inside the container*
 - *Work inside and outside the container interchangeably*
- *Others*
 - *Work offline*
 - *Hacking*

2.1 Projects

2.1.1 Unlimited projects

The number of projects you can add are so to speak unlimited. Simply add new project directories and they become automatically available in no time.

2.1.2 Automated virtual hosts

Creating a new project is literally done by creating a new directory on the file system. Everything else is automatically taken care of in the background. Virtual hosts are added instantly without having to restart any services.

2.1.3 Automated DNS records

The built-in DNS server will automatically make any DNS record available to your host system by using a wild-card DNS record.

2.1.4 Email catch-all

All outgoing emails originating from your projects are intercepted, stored locally and can be viewed within the bundled intranet. This removes the need to create developer DNS records in `/etc/hosts`.

2.1.5 Log files

Log files for every service are available. Either in the form of Docker logs or as actual log files mounted into the Devilbox git directory. The web and PHP server offer log files for each project separately.

2.1.6 Virtual host domains

Each of your virtual host will have its own domain. TLD can be freely chosen, such as `*.loc`, `*.local`, `*.com`, `*.org` or whatever you require.

2.2 Service and version choice

2.2.1 Selective start

Run only the Docker container you actually need, but be able to reload others on the fly once they are needed. So you could first startup PHP and MySQL only and in case you would require a Redis server you can attach it later to the Devilbox stack without having to restart anything.

2.2.2 Version choice

Each provided service (such as PHP, MySQL, PostgreSQL, etc) comes in many different versions. You can enable any combination that matches your perfect development stack.

2.2.3 LAMP and MEAN stack

Run a full LAMP stack with Apache or Nginx and even attach MEAN stack services such as MongoDB.

2.3 Configuration

2.3.1 Global configuration

All services can be configured globally by including your very own customized `php.ini`, `php-fpm.conf`, `my.cnf`, `nginx.conf`, `apache.conf` and other configuration files.

2.3.2 Version specific configuration

Each version of PHP can have its own `php.ini` and `php-fpm.conf` files, each version of MySQL, MariaDB or PerconaDB can have its own `my.cnf` files, each Apache..., each Nginx... you get the idea.

2.3.3 Project specific configuration

Even down to projects, the Devilbox allows for full customization when it comes to virtual host settings via `vhost-gen`.

2.4 Intranet

2.4.1 Command & Control Center

The intranet is your Command & Control Center showing you all applied settings, mount points, port exposures, hostnames and any errors including how they can be resolved.

2.4.2 Third-party tools

Mandatory web projects are also shipped: `phpMyAdmin`, `Adminer` and `OpcacheGui` as well as a web GUI to view all sent emails.

2.5 Dockerized

2.5.1 Portable

Docker container run on Linux, Windows and MacOS, so does the Devilbox. This ensures that no matter what operating system you are currently on, you can always run your development stack.

2.5.2 Built nightly

Docker images (at least official Devilbox Docker images) are built nightly and pushed to Dockerhub to ensure to always have the latest versions installed and be up-to-date with any security patches that are available.

2.5.3 Ships popular development tools

The Devilbox is also designed to be a development environment offering many tools used for everyday web development, no matter if frontend or backend.

2.5.4 Work inside the container

Instead of working on you host operating system, you can do everything inside the container. This allows you to have all tools pre-installed and a working unix environment ready.

2.5.5 Work inside and outside the container interchangeably

No matter if you work on your host operating system or inside the Docker container. Special mount points and port-forwards are already in place to make both look the same to you.

2.6 Others

2.6.1 Work offline

The Devilbox only requires internet initially to pull the required Docker images, once this is done you can work completely offline. No need for an active internet connection.

2.6.2 Hacking

Last but not least, the Devilbox is bascially just a `docker-compose.yml` file and you can easily add any Docker images you are currently missing in the Devilbox setup.

CHAPTER 3

Install the Devilbox

Important:

Read first Ensure you have read this document to understand how this documentation works.

Table of Contents

- *Supported OS*
- *Requirements*
- *Download the devilbox*
 - *Checkout a different release*
- *Create .env file*
- *Adjust .env file*
 - *Find your user id*
 - *Find your group id*
- *Checklist*

3.1 Supported OS

The devilbox runs on all operating systems that provide Docker and Docker Compose.



3.2 Requirements

The only requirements for the devilbox is to have `Docker` and `Docker Compose` installed, everything else is bundled and provided withing the Docker container. The minimum required versions are listed below:

- `Docker`: 1.12.0+
- `Docker Compose`: 1.9.0+

Additionally you will require `git` in order to clone the devilbox project.

Warning:

Docker Toolbox Use **native Docker** and do not use the **Docker Toolbox**. If you still have to use the Docker Toolbox (e.g. for Windows 7 or older Macs) read up on this section.

Warning: Docker itself requires super user privileges which is granted to a system wide group called `docker`. After having installed Docker on your system, ensure that your local user is assigned to the `docker` group. Check this via `groups` or `id` command.

See also:

Install Docker Have a look at this page to help you install `Docker` for your operating system.

Install Docker Compose Have a look at this page to help you install `Docker Compose` for your operating system.

3.3 Download the devilbox

The devilbox does not need to be installed. The only thing that is required is its `git` directory. To download that, open a terminal and copy/paste the following command.

```
host> git clone https://github.com/cytopia/devilbox
```

3.3.1 Checkout a different release

You now have the devilbox downloaded at the latest version (`git master branch`). This is also recommended as it receives bugfixes frequently. If you however want to stay on a stable release, you need to check out s specific `git` tag.

Lets say you want your devilbox setup to be at release `0.12.1`, all you have to do is to check out this specific `git` tag.

```
host> cd path/to/devilbox
host> git checkout 0.12.1
```

Warning: Whenever you check out a different version, make sure that your `.env` file is up-to-date with the bundled `env-example` file. Different Devilbox releases might require different settings to be available inside the `.env` file. Refer to the next section for how to create the `.env` file.

3.4 Create `.env` file

Inside the cloned devilbox git directory, you will find a file called `env-example`. This file acts as a template with sane defaults for `Docker Compose`. In order to use it, it must be copied to a file named `.env`. (Note the leading dot).

```
host> cp env-example .env
```

The `.env` file does nothing else then providing environment variables for `Docker Compose` and in this case it is used as the main configuration file for the devilbox by providing all kinds of settings (such as which version to start up).

See also:

Docker Compose env file Official Docker documentation about the `.env` file

.env file All available Devilbox `.env` values and their description

3.5 Adjust `.env` file

To get you started, there are only two variables that need to be adjusted:

- `NEW_UID`
- `NEW_GID`

The values for those two variables refer to your local (on your host operating system) user id and group id. To find out what the values are required in your case, issue the following commands on a terminal:

3.5.1 Find your user id

```
host> id -u
```

3.5.2 Find your group id

```
host> id -g
```

In most cases both values will be 1000, but for the sake of this example, let's assume a value of 1001 for the user id and 1002 for the group id.

Open the `.env` file with your favorite text editor and adjust those values:

Listing 1: .env

```
host> vi .env  
NEW_UID=1001  
NEW_GID=1002
```

Warning: Make sure that you use the values provided by `id -u` and `id -g`.

See also:

Synchronize container permissions Read up more on the general problem of trying to have synchronized permissions between the host system and a running Docker container.

3.6 Checklist

1. Docker and Docker Compose are installed at minimum required version
2. Your user is part of the docker group
3. Devilbox is cloned
4. .env file is created
5. User and group id have been set in .env file

That's it, you have finished the first section and have a working Devilbox ready to be started.

Update the Devilbox

If you are in the initial install process, you can safely skip this section and come back once you actually want to update the Devilbox.

Table of Contents

- *Update git repository*
 - *Stop container*
 - *Case 1: Update master branch*
 - *Case 2: Checkout release tag*
 - *Keep .env file in sync*
 - *Recreate container*
- *Update Docker images*
 - *Update one Docker image*
 - *Update all currently set Docker images*
 - *Update all available Docker images for all versions*
- *Checklist git repository*
- *Checklist Docker images*

4.1 Update git repository

4.1.1 Stop container

Before updating your git branch or checking out a different tag or commit, make sure to properly stop all devilbox containers:

```
# Stop containers
host> cd path/to/devilbox
host> docker-compose stop

# Ensure containers are stopped
host> docker-compose ps
```

4.1.2 Case 1: Update master branch

If you simply want to update the master branch, do a `git pull origin master`:

```
# Update master branch
host> cd path/to/devilbox
host> git pull origin master
```

4.1.3 Case 2: Checkout release tag

If you want to checkout a specific release tag (such as `0.12.1`), do a `git checkout 0.12.1`:

```
# Checkout release
host> cd path/to/devilbox
host> git checkout 0.12.1
```

4.1.4 Keep `.env` file in sync

Warning: Whenever you check out a different version, make sure that your `.env` file is up-to-date with the bundled `env-example` file. Different Devilbox releases might require different settings to be available inside the `.env` file.

You can also compare your current `.env` file with the provided `env-example` file by using your favorite diff editor:

```
host> vimdiff .env env-example
```

```
host> diff .env env-example
```

```
host> meld .env env-example
```

4.1.5 Recreate container

Whenever the path of a volume changes (either due to upstream changes in git or due to you changing it manually in the `.env` file) you need to remove the stopped container and have them fully recreated during the next start.

```
# Remove anonymous volumes
host> cd path/to/devilbox
host> docker-compose rm
```

See also:

Remove stopped container

4.2 Update Docker images

Updating the git branch shouldn't be needed to often, most changes are actually shipped via newer Docker images, so you should frequently update those.

This is usually achieved by issuing a `docker pull` command with the correct image name and image version or `docker-compose pull` for all currently selected images in `.env` file. For your convenience there is a shell script in the Devilbox git directory: `update-docker.sh` which will update all available Docker images at once for every version.

Note: The Devilbox own Docker images (Apache, Nginx, PHP and MySQL) are even built every night to ensure latest security patches and tool versions are applied.

4.2.1 Update one Docker image

Updating or pulling a single Docker image is accomplished by `docker pull <image>:<tag>`. This is not very handy as it is quite troublesome to do it separately per Docker image.

You first need to find out the image name and then also the currently used image tag.

```
host> grep 'image:' docker-compose.yml

image: cytopia/bind:0.11
image: devilbox/php-fpm:${PHP_SERVER:-7.0}-work
image: devilbox/${HTTPD_SERVER:-nginx-stable}:0.13
image: cytopia/${MYSQL_SERVER:-mariadb-10.1}:latest
image: postgres:${PGSQL_SERVER:-9.6}
image: redis:${REDIS_SERVER:-3.2}
image: memcached:${MEMCD_SERVER:-latest}
image: mongo:${MONGO_SERVER:-latest}
```

After having found the possible candidates, you will still have to find the corresponding value inside the `.env` file. Let's do it for the PHP image:

```
host> grep '^PHP_SERVER' .env

PHP_SERVER=5.6
```

So now you can substitute the `${PHP_SERVER}` variable from the first command with `5.6` and finally pull a newer version:

```
host> docker pull devilbox/php-fpm:5.6-work
```

Not very efficient.

4.2.2 Update all currently set Docker images

This approach is using `docker-compose pull` to update all images, but only for the versions that are actually set in `.env`.

```
host> docker-compose pull

Pulling bind (cytopia/bind:0.11)...
Pulling php (devilbox/php-fpm:5.6-work)...
Pulling httpd (devilbox/apache-2.2:0.13)...
Pulling mysql (cytopia/mysql-5.7:latest)...
Pulling pgsql (postgres:9.6)...
Pulling redis (redis:4.0)...
Pulling memcd (memcached:1.5.2)...
Pulling mongo (mongo:3.0)...
```

This is most likely the variant you want.

4.2.3 Update all available Docker images for all versions

In case you also want to pull/update every single of every available Devilbox image, you can use the provided shell script, which has all versions hardcoded and pulls them for you:

```
host> ./update-docker.sh
```

4.3 Checklist git repository

1. Ensure containers are stopped and removed/recreated (`docker-compose stop && docker-compose rm`)
2. Ensure desired branch, tag or commit is checked out or latest changes are pulled
3. Ensure `.env` file is in sync with `env-example` file

4.4 Checklist Docker images

1. Ensure `docker-compose pull` or `./update-docker.sh` is executed

CHAPTER 5

Start the Devilbox

Congratulations, when you have reached this page everything has been set up and you can now get your hands dirty.

Note: Starting and stopping containers is done via `docker-compose`. If you have never worked with it before, have a look at their documentation for an [overview](#), [up](#) and [stop](#) commands.

Table of Contents

- *Start all container*
- *Start some container*
- *Open Devilbox intranet*
- *Checklist*

5.1 Start all container

If you want all provided services to be available (as defined in `docker-compose.yml`), just start them all via:

```
host> docker-compose up
```

- If you want to gracefully stop all container, hit `Ctrl + c`
- If you want to kill all container, hit `Ctrl + c` twice

5.2 Start some container

If you don't require all services to be up and running and let's say just PHP, HTTPD and MYSQL, enter the following command:

```
host> docker-compose up httpd php mysql
```

- If you want to gracefully stop all started container, hit `Ctrl + c`
- If you want to kill all started container, hit `Ctrl + c` twice

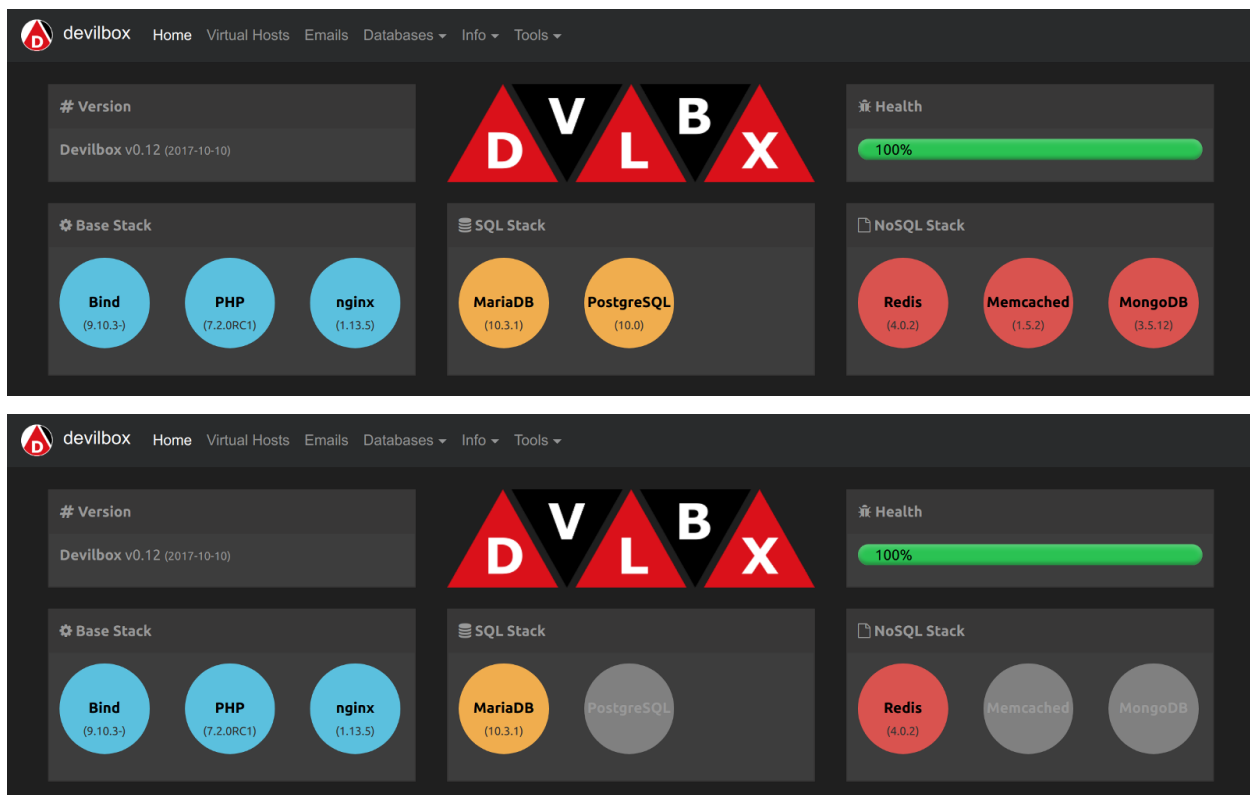
See also:

Available container Have a look at this page to get an overview about all available container and by what name they have to be specified.

5.3 Open Devilbox intranet

Once `docker-compose up` has finished and all or the selected container are up and running, you can visit the Devilbox intranet with your favorite Web browser at <http://localhost> or <http://127.0.0.1>.

The Intranet start page will also show you all running and failed containers:



Warning:

Docker Toolbox When you are using Docker Toolbox the Devilbox Web server port will not be available on your host computer. You have to forward the virtual machines port to your host computer. Read more about it on this guide.

5.4 Checklist

1. Docker container are started successfully with `docker-compose up`
2. Intranet is reachable via `http://localhost` or `http://127.0.0.1`

CHAPTER 6

Directory overview

Important: The directory overview only provides you some theoretical, but useful insights about how it all works together. You should at least read it once to be able to debug any problems you might encounter.

If you have read it already, jump directly to *Create your first project*

Table of Contents

- *Data directory*
- *Project directory*
- *Docroot directory*
- *Domain suffix*
- *Making sense of it*
- *Checklist*

6.1 Data directory

By default all your projects must be created in the `./data/www/` directory which is inside in your Devilbox git directory. This can be changed as well, but is outside the scope of this *getting started tutorial*.

You can verify that the path is actually `./data/www/` by checking your `.env` file:

```
host> grep HTTPD_DATADIR .env  
HOST_PATH_HTTPD_DATADIR=./data/www
```

6.2 Project directory

The project directory is a directory directly within the data directory.

This represents one project.

By creating this directory, the web server will create a new virtual host for you. This happens fully automated and there is nothing else required to do except just to create a directory.

The name of this directory will also be used to build up the final project url together with the domain suffix: `http://<project directory>.<domain suffix>`

Create as many project directories as you require.

6.3 Docroot directory

The docroot directory is a directory within each project directory from which the webserver will serve the files.

By default this directory must be named `htdocs`. This can be changed as well, but is outside the scope of this *getting started tutorial*.

You can verify that the docroot directory is actually `htdocs` by checking your `.env` file:

```
host> grep DOCROOT_DIR .env

HTTPD_DOCROOT_DIR=htdocs
```

6.4 Domain suffix

The default domain suffix (`TLD_SUFFIX` variable in `.env` file) is `loc`. That means that all your projects will be available under the following address: `http://<project-directory>.loc`. This can be changed as well, but is outside the scope of this *getting started tutorial*.

You can verify that the suffix is actually `loc` by checking your `.env` file:

```
host> grep ^TLD_SUFFIX .env

TLD_SUFFIX=loc
```

6.5 Making sense of it

Ok, let's sum it up and make sense of the previously provided information. To better illustrate the behaviour we are going to use `project-1` as our project directory name.

| Item | Example | Description |
|---------------|--|---|
| data dir | <code>./data/www</code> | Where all of your projects reside. |
| project dir | <code>./data/www/project-1</code> | A single project. It's name will be used to create the url. |
| docroot dir | <code>./data/www/project-1/htdocs</code> | Where the webserver looks for files within your project. |
| domain suffix | <code>loc</code> | Suffix to build up your project url. |
| project url | <code>http://project-1.loc</code> | Final resulting project url. |

data dir

This directory is mounted into the `httpd` and `php` container, so that both know where all projects can be found. This is also the place where you create `project` directories for each of your projects.

project dir

Is your project and used to generate the virtual host together with the domain suffix.

docroot dir

A directory inside your `project` dir from where the webserver will actually serve your files.

domain suffix

Used as part of the project url.

6.6 Checklist

1. You know what the data directory is
2. You know what the project directory is
3. You know what the docroot directory is
4. You know what the domain suffix is
5. You know how domains are constructed

CHAPTER 7

Create your first project

Important: Ensure you have read *Directory overview* to understand what is going on under the hood.

Note: This section not only applies for one project, it applied for as many projects as you need. **There is no limit in the number of projects.**

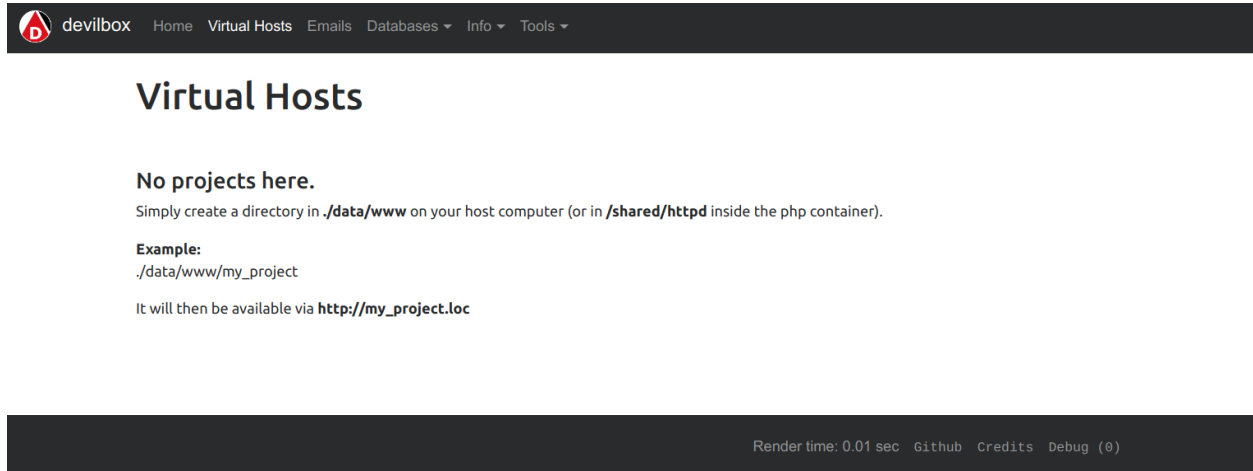
Table of Contents

- *Step 1: visit Intranet vhost page*
- *Step 2: create a project directory*
- *Step 3: create a docroot directory*
- *Step 4: create a DNS entry*
 - *Add DNS for Linux and MacOS (native Docker)*
 - *Add DNS for Windows (native Docker)*
 - *Add DNS for Docker Toolbox*
 - *Back to intranet*
- *Step 5: Visit your project*
- *Step 6: Create a hello world*
- *Checklist*

7.1 Step 1: visit Intranet vhost page

Before starting, have a look at the vhost page at <http://localhost/vhosts.php>

It should look like the screenshot below and will actually already provide the information needed to create a new project.



7.2 Step 2: create a project directory

In your Devilbox git directory, navigate to `./data/www` and create a new directory.


Note: Choose the directory name wisely, as it will be part of the domain for that project. For this example we will use `project-1` as our project name.

```
# navigate to your Devilbox git directory
host> cd path/to devilbox

# navigate to the data directory
host> cd data/www

# create a new project directory named: project-1
host> mkdir project-1
```

Visit the vhost page again and see what has changed: <http://localhost/vhosts.php>


Home Virtual Hosts Emails Databases Info Tools

Virtual Hosts

| Project | DocumentRoot | Valid | URL |
|-----------|-----------------------------|-------|---|
| project-1 | ./data/www/project-1/htdocs | ERR | error Missing htdocs directory in: ./data/www/project-1/ |

Render time: 0.01 sec Github Credits Debug (0)

So what has happened?

By having created a project directory, the web server container has created a new virtual host. However it has noticed, that the actual document root directory does not yet exist and therefore it cannot serve any files yet.

7.3 Step 3: create a docroot directory

Note: As described in *Docroot directory* the docroot directory name must be `htdocs` for now.


Navigate to your newly created project directory and create a directory named *htdocs* inside it.

```
# navigate to your Devilbox git directory
host> cd path/to devilbox

# navigate to your above created project directory
host> cd data/www/project-1

# create the docroot directory
host> mkdir htdocs
```

Visit the vhost page again and see what has changed: <http://localhost/vhosts.php>


Home Virtual Hosts Emails Databases Info Tools

Virtual Hosts

| Project | DocumentRoot | Valid | URL |
|-----------|-----------------------------|-------|--|
| project-1 | ./data/www/project-1/htdocs | ERR | No Host DNS record found. Add the following to /etc/hosts : 127.0.0.1 project-1.loc |

Render time: 0.01 sec Github Credits Debug (0)

So what has happened?

By having created the docroot directory, the web server is now able to serve your files. However it has noticed, that you have no way yet, to actually visit your project url, as no DNS record for it exists yet.

The intranet already gives you the exact string that you can simply copy into your `/etc/hosts` file on your host operating system to solve this issue.

Important: This will only work on **native Docker** for Linux or MacOS. Read up on the next section to also find out how to do that on **Docker Toolbox** and Windows.

7.4 Step 4: create a DNS entry

Note: This step can also be automated via the bundled DNS server to automatically provide catch-all DNS entries to your host computer, but is outside the scope of this *getting started tutorial*.

7.4.1 Add DNS for Linux and MacOS (native Docker)

On Linux and MacOS (when using the native Docker), this step is fairly simple. The intranet provides you the exact string you need to paste into your `/etc/hosts` file on your host operating system.

```
# Open your /etc/hosts file with sudo or root privileges
# and add the following DNS entry
host> sudo vi /etc/hosts

127.0.0.1 project-1.loc
```

7.4.2 Add DNS for Windows (native Docker)

On Windows (when using the native Docker), you can also copy paste the command provided by the intranet, however the destination file is different. You have to add this string into: `C:\Windows\System32\drivers\etc`.

Open `C:\Windows\System32\drivers\etc` with administrative privileges and add the following entry

```
127.0.0.1 project-1.loc
```

7.4.3 Add DNS for Docker Toolbox

When using `Docker Toolbox` the Devilbox runs inside a virtual machine and therefore the Webserver port (80) is not exposed to your host operating system. So your DNS record must point to the virtual machine instead of your host system.

1. Find out the IP address the virtual machine is running on
2. Add a DNS entry to your host operating system for this IP address.

For the sake of this example, let's assume the virtual machine is running on `192.16.0.1`, then the DNS record you will have to add instead on your host operating system is:

Docker Toolbox on MacOS

```
host> sudo vi /etc/hosts  
192.16.0.1 project-1.loc
```


Docker Toolbox on Windows

Open `C:\Windows\System32\drivers\etc` with administrative privileges and add the following entry

```
192.16.0.1 project-1.loc
```

7.4.4 Back to intranet

Visit the vhost page again and see what has changed: <http://localhost/vhosts.php>

 devilbox [Home](#) [Virtual Hosts](#) [Emails](#) [Databases](#) [Info](#) [Tools](#)

Virtual Hosts

| Project | DocumentRoot | Valid | URL |
|-----------|-----------------------------|-------|--|
| project-1 | ./data/www/project-1/htdocs | OK | project-1.loc |

Render time: 0.01 sec [Github](#) [Credits](#) [Debug \(0\)](#)

So what has happened?

By having created the DNS record, the Devilbox intranet is aware that everything is setup now and gives you a link to your new project.

7.5 Step 5: Visit your project

On the intranet, click on your project link. This will open your project in a new Browser tab or visit <http://project-1.loc>

403 Forbidden

nginx/1.12.1

So what has happened?

Everything is setup now, however the webserver is trying to find a `index.php` file in your document root which does not yet exist.

So all is left for you to do is to add your HTML or PHP files.

7.6 Step 6: Create a hello world

Navigate to your docroot directory within your project and create a `index.php` file with some output.

```
# navigate to your Devilbox git directory
host> cd path/to devilbox

# navigate to your projects docroot directory
host> cd data/www/project-1/htdocs

# Create a hello world index.php file
host> echo "<?php echo 'hello world';" > index.php
```

Alternatively create an `index.php` file in `data/www/project-1/htdocs` with the following contents:

```
<?php echo 'hello world';
```

Visit your project url again and see what has changed: <http://project-1.loc>

hello world

7.7 Checklist

1. Project directory is created
2. Docroot directory is created
3. DNS entry is added to the host operating system
4. PHP files are added to your docroot directory

CHAPTER 8

Read log files

The logging behaviour is determined by the value of *DOCKER_LOGS* inside your `.env` file. By default logs are mounted to the host operating system for convenient access.

Table of Contents

- *Mounted logs*
- *Docker logs*
- *Checklist*

8.1 Mounted logs

By default log files for PHP, the webserver and the MySQL server are mounted to the host system into your Devilbox git directory under `./log/`. All logs are separated by service version in the following format: `./log/<service>-<version>/`

The log directory structure would look something like this:

```
host> cd path/to/devilbox
host> tree log

log/
├── nginx-stable/
│   ├── nginx-stable/
│   ├── defaultlocalhost-access.log
│   ├── defaultlocalhost-error.log
│   ├── <project-name>-access.log      # Each project has its own access log
│   └── <project-name>-error.log      # Each project has its own error log
└── mariadb-10.1/
    └── error.log
```

(continues on next page)

(continued from previous page)

```
|   ├── query.log
|   ├── slow.log
|   └── php-fpm-7.1/
|       ├── php-fpm.access
|       └── php-fpm.error
```

Use your favorite tools to view log files such as `tail`, `less`, `more`, `cat` or others.

Important: Currently logs are only mounted for PHP, HTTPD and MYSQL container. All other services will log to Docker logs.

8.2 Docker logs

You can also change the behaviour where logs are streamed by setting `DOCKER_LOGS` to 1 inside your `.env` file. When doing logs are sent to Docker logs.

When using this approach, you need to use the `docker-compose logs` command to view your log files from within the Devilbox git directory.

```
host> cd path/to/devilbox
host> docker-compose logs
```

When you want to continuously watch the log output (such as `tail -f`), you need to append `-f` to the command.

```
host> cd path/to/devilbox
host> docker-compose logs -f
```

When you only want to have logs displayed for a single service, you can also append the service name (works with or without `-f` as well):

```
host> cd path/to/devilbox
host> docker-compose logs php -f
```

Important: This currently does not work for the MySQL container, which will always log to file.

8.3 Checklist


1. You know how to switch between file and Docker logs
2. You know where log files are mounted
3. You know how to access Docker logs

CHAPTER 9

Email catch-all

All your projects can send emails to whatever recipient. You do not have to worry that they will actually being sent. Each PHP container runs a local postfix mailserver that intercepts all outgoing mails and puts them into a local mailbox by the user `devilbox`.

In order to view sent emails open up the devilbox intranet <http://localhost/mail.php>. There you can also test email sending and verify that they really stay locally.


devilbox

[Home](#)
[Virtual Hosts](#)
[Emails](#)
[Databases](#)
[Info](#)
[Tools](#)

Mail

Send test Email

Received Emails

| # | Date | From | To | Subject |
|---|---------------------|-----------------------|---------------------|-------------------------|
| 3 | 14:30 2018-04-02 | devilbox@851826c075c1 | john@example.com | Your forum registration |
| 2 | 14:30 2018-04-02 | devilbox@851826c075c1 | noreply@example.com | Automated reply message |
| 1 | 14:29 2018-04-02 | devilbox@851826c075c1 | all@company.org | Test Email |

This email was intended to go to company.org

```

From devilbox@851826c075c1 Mon Apr 2 14:29:44 2018
Return-Path:
X-Original-To: all@company.org
Delivered-To: devilbox@851826c075c1
Received: by 851826c075c1 (Postfix, from userid 1000)
        id AEA6136027E; Mon, 2 Apr 2018 12:29:44 +0000 (UTC)
To: all@company.org
Subject: Test Email
X-PHP-Originating-Script: 1000:mail.php
Message-Id: <20180402122944.AEA6136027E@851826c075c1>
Date: Mon, 2 Apr 2018 12:29:44 +0000 (UTC)
From: devilbox@851826c075c1

This email was intended to go to company.org

```

In the above image from the intranet you can see that all emails sent to whatever recipient have been caught by the Devilbox and are available to be read.

CHAPTER 10

Enter the PHP container

Another core feature of the Devilbox is, to be totally independent of what you have or have not installed on your host operating system.

The Devilbox already ships with many common developer tools which are installed inside each PHP container, so why not make use of it.

The only thing you might need to install on your host operating system is your favourite IDE or editor to actually start coding.

See also:

If you want to find out what tools are available inside the PHP container, visit the following section: [Available tools](#).

Table of Contents

- *How to enter*
 - *Linux and MacOS*
 - *Windows*
- *How to become root*
- *Tools*
 - *What is available*
 - *How to update them*
- *Advanced*
- *Checklist*

10.1 How to enter

Note: You can only enter the PHP container if it is running.

10.1.1 Linux and MacOS

On Linux and MacOS you can simply execute the provided shell script: `shell.sh`. By doing so it will enter you into the PHP container and bring you to `/shared/httpd`.

```
# Execute on the host operating system
host> ./shell.sh

# Now you are inside the PHP Linux container
devilbox@php-7.0.19 in /shared/httpd $
```

10.1.2 Windows

On Windows you have a different script to enter the PHP container: `shell.bat`. Just run it and it will enter you into the PHP container and bring you to `/shared/httpd`.

```
# Execute on the host operating system
C:/Users/user1/devilbox> shell.bat

# Now you are inside the PHP Linux container
devilbox@php-7.0.19 in /shared/httpd $
```

10.2 How to become root

When you enter the container with the provided scripts, you are doing so as the user `devilbox`. If you do need to perform any actions as root (such as installing new software), you can use the password-less `sudo`.

```
# Inside the PHP Linux container as user devilbox
devilbox@php-7.0.19 in /shared/httpd $ sudo su -

# Now you are root and can do anything you want
root@php-7.0.19 in /shared/httpd $
```

Note: As this action is inside a Docker container, there is no difference between Linux, MacOS or Windows. Every host operating system is using the same Docker container - equal across all platforms.

10.3 Tools

10.3.1 What is available

There are lots of tools available, for a full overview see [Available tools](#). If you think you are missing a tool, install it yourself as root, or open up an issue on github to get it backed into the Docker image permanently.

See also:

Available tools

10.3.2 How to update them

There is no need to update the tools itself. All Docker images are rebuilt every night and automatically pushed to Docker hub to ensure versions are outdated at a maximum of 24 hours.

The only thing you have to do, is to update the Docker images itself, simply by pulling a new version.

See also:

Update Docker images

10.4 Advanced

This is just a short overview about the possibility to work inside the container. If you want to dig deeper into this topic there is also a more advanced tutorial available:

See also:

Work inside the container

10.5 Checklist

- You know how to enter the PHP container on Linux, MacOS or Windows
- You know how to become `root` inside the PHP container
- You know what tools are available inside the PHP container
- You know how to update the tools by pulling new versions of the Docker images

CHAPTER 11

The Intranet

The intranet is your command & control center showing all kinds of information and settings currently in effect. It also offers third-party projects to do all sorts of database manipulation.

Table of Contents

- *Devilbox tools*
 - *Overview*
 - *Virtual hosts*
 - *Emails*
 - *Databases*
 - *Info pages*
- *Third-party tools*
 - *phpMyAdmin*
 - *Adminer*
 - *OpcacheGUI*
- *Settings*
 - *Password protect the intranet*
 - *Disable the intranet*
- *Checklist*

11.1 Devilbox tools

11.1.1 Overview

The start page is there to check if everything works as expected. It shows all desired Docker containers you wanted to start and if they succeeded, as well as their ports, mount points and special settings applied via `.env`.


[Home](#)
[Virtual Hosts](#)
[Emails](#)
[Databases](#)
[Info](#)
[Tools](#)

Version

Devilbox v0.9 (2017-05-20)



Health

100%

Base Stack

Bind (9.9.5-9)

PHP (7.0.19)

nginx (1.12.0)

SQL Stack

MariaDB (10.1.23)

PostgreSQL (9.6.3)

NoSQL Stack

Redis (3.2.8)

Memcached (1.4.21)

mongodb

PHP Container Setup

You can also enter the php container and work from inside. The following is available inside the container:

| Settings | |
|---------------|---------------|
| uid | 1001 |
| gid | 1001 |
| vHost TLD | *.loc |
| DNS | Enabled |
| Postfix | Enabled |
| Xdebug | Yes |
| Xdebug Remote | 192.168.0.215 |
| Xdebug Port | 9000 |

| Tools | |
|---------------|---------------|
| composer | 1.4.2 |
| drush | 8.1.11 |
| drush-console | not installed |
| git | 1.8.3.1 |
| node | 6.10.2 |
| npm | 3.10.10 |

PHP Container Status

The PHP Docker can connect to the following services via the specified hostnames and IP addresses.

| Service | Hostname / IP |
|-------------------|--|
| Httpd connect | <input checked="" type="checkbox"/> httpd |
| | <input checked="" type="checkbox"/> 172.16.238.11 |
| | <input checked="" type="checkbox"/> random.loc |
| MySQL connect | <input checked="" type="checkbox"/> mysql |
| | <input checked="" type="checkbox"/> 172.16.238.12 |
| | <input checked="" type="checkbox"/> 127.0.0.1 |
| PgSQL connect | <input checked="" type="checkbox"/> pgsql |
| | <input checked="" type="checkbox"/> 172.16.238.13 |
| | <input checked="" type="checkbox"/> 127.0.0.1 |
| Redis connect | <input checked="" type="checkbox"/> redis |
| | <input checked="" type="checkbox"/> 172.16.238.14 |
| | <input checked="" type="checkbox"/> 127.0.0.1 |
| Memcached connect | <input checked="" type="checkbox"/> memcd |
| | <input checked="" type="checkbox"/> 172.16.238.15 |
| | <input checked="" type="checkbox"/> 127.0.0.1 |
| Bind connect | <input checked="" type="checkbox"/> bind |
| | <input checked="" type="checkbox"/> 172.16.238.100 |

Networking

| Docker | Hostname | IP |
|-----------|----------|----------------|
| php | php | 172.16.238.10 |
| httpd | httpd | 172.16.238.11 |
| mysql | mysql | 172.16.238.12 |
| pgsql | pgsql | 172.16.238.13 |
| redis | redis | 172.16.238.14 |
| memcached | memcd | 172.16.238.15 |
| bind | bind | 172.16.238.100 |

Ports

| Docker | Host port | Docker port |
|-----------|------------------|-------------|
| php | - | 9000 |
| httpd | 127.0.0.1:80 | 80 |
| mysql | 127.0.0.1:3306 | 3306 |
| pgsql | 127.0.0.1:5432 | 5432 |
| redis | 127.0.0.1:6379 | 6379 |
| memcached | 127.0.0.1:11211 | 11211 |
| bind | 127.0.0.1:53/tcp | 53/tcp |
| | 127.0.0.1:53/udp | 53/udp |

Data mounts

| Docker | Host path | Docker path |
|-----------|--------------------------|---------------------------------|
| php | /data/www | /shared/httpd |
| httpd | /data/www | /shared/httpd |
| mysql | /data/mysql/mariadb-10.1 | /var/lib/mysql |
| pgsql | /data/pgsql/9.6 | /var/lib/postgresql/data/pgdata |
| redis | - | - |
| memcached | - | - |
| bind | - | - |

Config mounts

| Docker | Host path | Docker path |
|-----------|-------------------|-------------------|
| php | /cfg/php-fpm-7.0 | /etc/php-custom.d |
| httpd | - | - |
| mysql | /cfg/mariadb-10.1 | /etc/mysql/conf.d |
| pgsql | - | - |
| redis | - | - |
| memcached | - | - |
| bind | - | - |

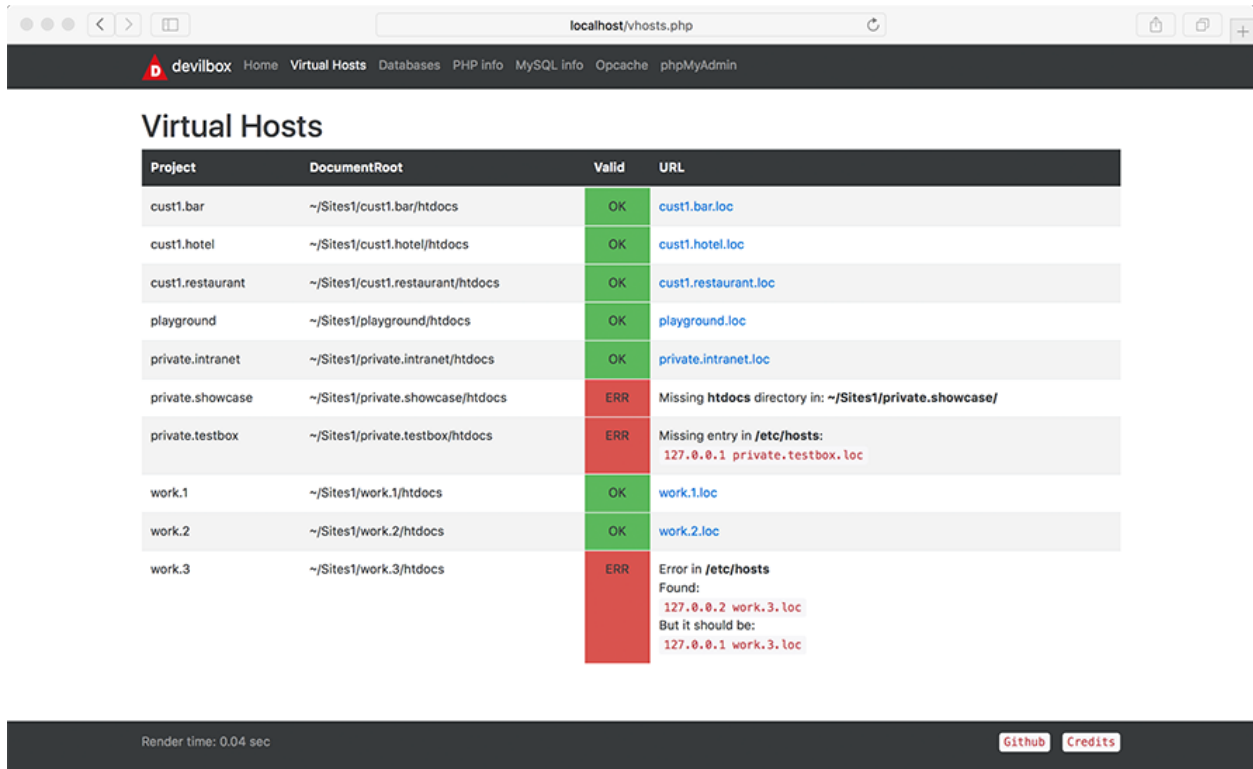
Log mounts

| Docker | Host path | Docker path |
|-----------|-----------------------|-----------------------|
| php | /log/php-fpm-7.0 | /var/log/php |
| httpd | /log/nginx-stable | /var/log/nginx-stable |
| mysql | /log/mariadb-10.1 | /var/log/mysql |
| pgsql | /log/pgsql-9.6 | /var/log/postgresql |
| redis | /log/redis-3.2 | /var/log/redis |
| memcached | /log/memcached-1.4.21 | /var/log/memcached |
| bind | - | - |

Render time: 1.07 sec
[Github](#)
[Credits](#)
[Debug \(0\)](#)

11.1.2 Virtual hosts

The virtual host page displays all available projects and let's you know if their configuration is correct, such as DNS settings or document root.



| Project | DocumentRoot | Valid | URL |
|------------------|----------------------------------|-------|---|
| cust1.bar | ~/Sites1/cust1.bar/htdocs | OK | cust1.bar.loc |
| cust1.hotel | ~/Sites1/cust1.hotel/htdocs | OK | cust1.hotel.loc |
| cust1.restaurant | ~/Sites1/cust1.restaurant/htdocs | OK | cust1.restaurant.loc |
| playground | ~/Sites1/playground/htdocs | OK | playground.loc |
| private.intranet | ~/Sites1/private.intranet/htdocs | OK | private.intranet.loc |
| private.showcase | ~/Sites1/private.showcase/htdocs | ERR | Missing htdocs directory in: ~/Sites1/private.showcase/ |
| private.testbox | ~/Sites1/private.testbox/htdocs | ERR | Missing entry in /etc/hosts : 127.0.0.1 private.testbox.loc |
| work.1 | ~/Sites1/work.1/htdocs | OK | work.1.loc |
| work.2 | ~/Sites1/work.2/htdocs | OK | work.2.loc |
| work.3 | ~/Sites1/work.3/htdocs | ERR | Error in /etc/hosts Found: 127.0.0.2 work.3.loc But it should be: 127.0.0.1 work.3.loc |

Render time: 0.04 sec

[Github](#) [Credits](#)

11.1.3 Emails

The email page displays all emails that would have been sent, but were caught by the integrated email catch-all functionality.

localhost/mail.php

devilbox

Home Virtual Hosts Databases Mail PHP Info MySQL Info Opcache phpMyAdmin

Mail

Send test Email

Received Emails

| # | Date | To | Subject |
|--|---------------------|---------------------------|------------------------|
| 3 | 12:19 2016-10-30 | test@example.com | Mailing to example.com |
| 2 | 12:19 2016-10-30 | root | Mail to root |
| 1 | 12:19 2016-10-30 | apache | Local delivery Test |
| <div> <div>From apache@7fcee044e148.localdomain Sun Oct 30 12:19:10 2016</div> <div>Return-Path:</div> <div>X-Original-To: apache</div> <div>Delivered-To: mailtrap@7fcee044e148.localdomain</div> <div>Received: by 7fcee044e148.localdomain (Postfix, from userid 48) id 64C09380; Sun, 30 Oct 2016 12:19:10 +0100 (CET)</div> <div>To: apache@7fcee044e148.localdomain</div> <div>Subject: Local delivery Test</div> <div>X-PHP-Originating-Script: 48:mail.php</div> <div>Message-Id: <20161030111910.64C09380@7fcee044e148.localdomain></div> <div>Date: Sun, 30 Oct 2016 12:19:10 +0100 (CET)</div> <div>From: apache@7fcee044e148.localdomain</div> <div>testing local users</div> </div> | | | |
| 0 | 12:18 2016-10-30 | cytopia@everythingcli.org | Testing Emails |

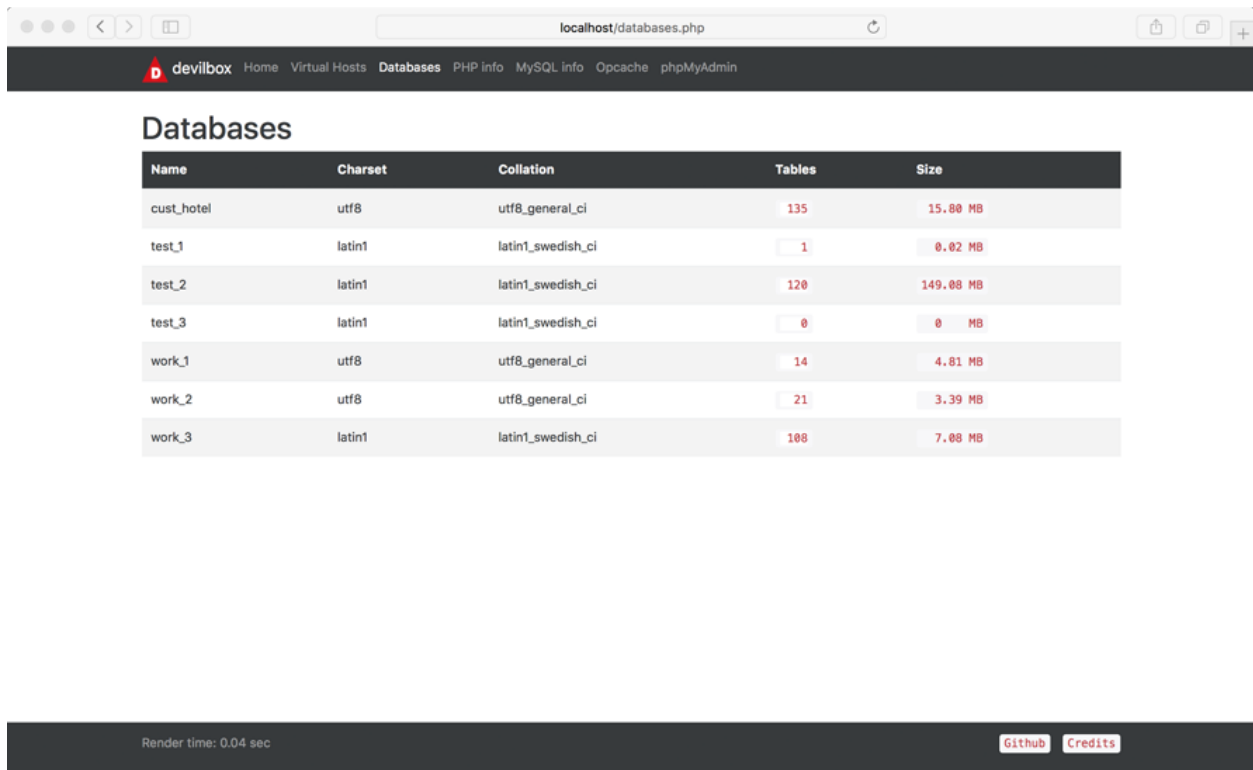
Render time: 0.02 sec

[Github](#)
[Credits](#)

11.1.4 Databases

There are several database pages for MySQL and NoSQL databases giving you an overview about what is currently in place, how many databases/schemas and or recors and what size they take up.

The following example shows the database page for MySQL:



The screenshot shows a web browser window with the address bar displaying 'localhost/databases.php'. The page title is 'Databases'. Below the title is a table with the following data:


| Name | Charset | Collation | Tables | Size |
|------------|---------|-------------------|--------|-----------|
| cust_hotel | utf8 | utf8_general_ci | 135 | 15.00 MB |
| test_1 | latin1 | latin1_swedish_ci | 1 | 0.02 MB |
| test_2 | latin1 | latin1_swedish_ci | 120 | 149.08 MB |
| test_3 | latin1 | latin1_swedish_ci | 0 | 0 MB |
| work_1 | utf8 | utf8_general_ci | 14 | 4.81 MB |
| work_2 | utf8 | utf8_general_ci | 21 | 3.39 MB |
| work_3 | latin1 | latin1_swedish_ci | 100 | 7.08 MB |

At the bottom of the page, there is a footer with the text 'Render time: 0.04 sec' and two links: 'Github' and 'Credits'.

11.1.5 Info pages

Info pages also exist for every Docker container which show various settings which are currently applied. The following example shows you the info page for PHP.

The following example shows you the info page for MySQL:


[Home](#)
[Virtual Hosts](#)
[Emails](#)
[Databases](#)
[Info](#)
[Tools](#)

MySQL Info

For reference see here:

- <https://dev.mysql.com/doc/refman/5.5/en/server-system-variables.html>
- <https://dev.mysql.com/doc/refman/5.6/en/server-system-variables.html>
- <https://dev.mysql.com/doc/refman/5.7/en/server-system-variables.html>

| Variable | Value |
|---|--------------|
| auto_increment_increment | 1 |
| auto_increment_offset | 1 |
| autocommit | ON |
| automatic_sp_privileges | ON |
| avoid_temporal_upgrade | OFF |
| back_log | 80 |
| basedir | /usr/ |
| big_tables | OFF |
| bind_address | 0.0.0.0 |
| binlog_cache_size | 32768 |
| binlog_checksum | CRC32 |
| binlog_direct_non_transactional_updates | OFF |
| binlog_error_action | ABORT_SERVER |
| binlog_format | ROW |
| binlog_group_commit_sync_delay | 0 |

11.2 Third-party tools

11.2.1 phpMyAdmin

phpMyAdmin is a free software tool written in PHP, intended to handle the administration of MySQL over the Web. phpMyAdmin supports a wide range of operations on MySQL and MariaDB. Frequently used operations (managing databases, tables, columns, relations, indexes, users, permissions, etc) can be performed via the user interface, while you still have the ability to directly execute any SQL statement.

11.2.2 Adminer

Adminer (formerly phpMinAdmin) is a full-featured database management tool written in PHP. Conversely to phpMyAdmin, it consist of a single file ready to deploy to the target server. Adminer is available for MySQL, MariaDB, PostgreSQL, SQLite, MS SQL, Oracle, Firebird, SimpleDB, Elasticsearch and MongoDB.

11.2.3 OpcacheGUI

OpcacheGui is a clean and responsive interface for Zend OPcache information, showing statistics, settings and cached files, and providing a real-time update for the information (using jQuery and React).

11.3 Settings

11.3.1 Password protect the intranet

If you share your projects over a LAN, but do not want anybody to view the Devilbox intranet, you can also password protect it.

See also:

In order to do so, have a look at the following `.env` variables:

- `DEVILBOX_UI_PROTECT`
- `DEVILBOX_UI_PASSWORD`

11.3.2 Disable the intranet

If you want a more production-like setup, you can also fully disable the Devilbox intranet. This is achieved internally by removing the default virtual host which serves the intranet. When the intranet is disabled, there is no way to access it.

See also:

In order to do so, have a look at the following `.env` variable:

- `DEVILBOX_UI_ENABLE`

11.4 Checklist

1. You know what tools are provided by the Devilbox intranet
2. You know how to password protect the Devilbox intranet
3. You know how to disable the Devilbox intranet

If you have already read all documents in the Getting started guide, you should be ready to fully operate the Devilbox. This section builds on top of that and gives you some best-practices as well as tips and tricks.

Table of Contents

- *Move data out of Devilbox directory*
 - *Projects*
 - *Databases*
 - * *MySQL*
 - * *PostgreSQL*
 - * *MongoDB*
 - *Version control .env file*
 - *Version control service config files*
- *PHP project hostname settings*
- *Timezone*

12.1 Move data out of Devilbox directory

One thing you should take into serious consideration is to move data such as your projects as well as persistent data of databases out of the Devilbox git directory.

The Devilbox git directory should be something that can be safely deleted and re-created without having to worry about losing any project data. There could also be the case that you have a dedicated hard-disk to store your projects or you have your own idea about a directory structure where you want to store your projects.

12.1.1 Projects

So let's assume all of your projects are already in place under `/home/user/workspace/web/`. Now you decide to use the Devilbox, but still want to keep your projects where they are at the moment.

All you have to do is to adjust the path of `HOST_PATH_HTTPD_DATADIR` in the `.env` file.

```
# Navigate to Devilbox git directory
host> cd path/to/devilbox

# Open the .env file with your favourite editor
host> vim .env
```

Now Adjust the value of `HOST_PATH_HTTPD_DATADIR`

Listing 1: `.env`

```
HOST_PATH_HTTPD_DATADIR=/home/user/workspace/web
```

That's it, whenever you start up the Devilbox `/home/user/workspace/web/` will be mounted into the PHP and the web server container into `/shared/httpd/`.

12.1.2 Databases

Moving your projects out of the Devilbox git directory is one step, you still need to take care about persistent data of all available databases as well.

Let's assume you desired location for database storage is at `/home/user/workspace/db/`.

MySQL

All you have to do is to adjust the path of `HOST_PATH_MYSQL_DATADIR` in the `.env` file.

```
# Navigate to Devilbox git directory
host> cd path/to/devilbox

# Open the .env file with your favourite editor
host> vim .env
```

Now Adjust the value of `HOST_PATH_MYSQL_DATADIR`

Listing 2: `.env`

```
HOST_PATH_MYSQL_DATADIR=/home/user/workspace/db/mysql
```

That's it, whenever you start up the Devilbox `/home/user/workspace/db/mysql/` will be mounted into the MySQL container.

PostgreSQL

All you have to do is to adjust the path of `HOST_PATH_PGSQL_DATADIR` in the `.env` file.

```
# Navigate to Devilbox git directory
host> cd path/to/devilbox

# Open the .env file with your favourite editor
host> vim .env
```

Now Adjust the value of *HOST_PATH_PGSQL_DATADIR*

Listing 3: .env

```
HOST_PATH_PGSQL_DATADIR=/home/user/workspace/db/pgsql
```

That's it, whenever you start up the Devilbox `/home/user/workspace/db/pgsql/` will be mounted into the PostgreSQL container.

MongoDB

All you have to do is to adjust the path of *HOST_PATH_MONGO_DATADIR* in the `.env` file.

```
# Navigate to Devilbox git directory
host> cd path/to/devilbox

# Open the .env file with your favourite editor
host> vim .env
```

Now Adjust the value of *HOST_PATH_MONGO_DATADIR*

Listing 4: .env

```
HOST_PATH_MONGO_DATADIR=/home/user/workspace/db/mongo
```

That's it, whenever you start up the Devilbox `/home/user/workspace/db/mongo/` will be mounted into the MongoDB container.

12.1.3 Version control `.env` file

The `.env` file is ignored by git, because this is *your* file to customize and it should be *your* responsibility to make sure to backup or version controlled.

One concept you can apply here is to have a separate **dotfiles** git repository. This is a repository that holds all of your configuration files such as vim, bash, zsh, xinit and many more. Those files are usually stored inside this repository and then symlinked to the correct location. By having all configuration files in one place, you can see and track changes easily as well as being able to jump back to previous configurations.

In case of the Devilbox `.env` file, just store this file in your repository and symlink it to the Devilbox git directory. This way you make sure that you keep your file, even when the Devilbox git directory is deleted and you also have a means of keeping track about changes you made.

You could also go further and have several `.env` files available somewhere. Each of those files holds different configurations e.g. for different projects or customers.

- `env-customer1`
- `env-php55`
- `env-project3`

You would then simply symlink one of those files to the Devilbox git directory.

12.1.4 Version control service config files

Todo: This will require some changes on the Devilbox and will be implemented shortly.

- Symlink and have your own git directory
- Separate data partition, backups

12.2 PHP project hostname settings

When configuring your PHP projects to use MySQL, PostgreSQL, Redis, Mongo and other services, make sure to set the hostname of each of those services to `127.0.0.1`.

Why is that?

The PHP container port-forwards each service port to its own listen address on `127.0.0.1`. The Devilbox also exposes each of those service ports to the host operating system on `127.0.0.1`.

This allows you to keep your project configuration unchanged and have the same behaviour inside the PHP container and on your host operating system.

Important: Do not mix up `localhost` with `127.0.0.1`. They behave differently! Use `127.0.0.1` and do not use `localhost`.

As an example, if you want to access the MySQL database from within the PHP container, you do the following:

```
# Navigate to Devilbox git directory
host> cd path/to/devilbox

# Enter the PHP container
host> ./shell.sh

# Enter the MySQL console
php> mysql -u root -h 127.0.0.1 -p
mysql>
```

The very same command applies to access the MySQL database from your host operating system:

```
# Enter the MySQL console
host> mysql -u root -h 127.0.0.1 -p
mysql>
```

So no matter if you use the Devilbox or have another LAMP stack installed locally on your host operating system, you do not have to change your configuration files if you stick to this tip.

So any of your projects php files that configure MySQL as an example should point the hostname or IP address of the MySQL server to `127.0.0.1`:

```
<?php
// MySQL server connection in your project configuration
mysql_host = '127.0.0.1';
mysql_port = '3306';
mysql_user = 'someusername';
mysql_pass = 'somepassword';
?>
```

See also:

Work inside the container

12.3 Timezone

The *TIMEZONE* value will affect PHP, web server and MySQL container equally. It does however not affect any other official Docker container that are used within the Devilbox. This is an issue that is currently still being worked on.

Feel free to change this to any timezone you require for PHP and MySQL, but keep in mind that timezone values for databases can be painful, once you want to switch to a different timezone.

A good practice is to always use UTC on databases and have your front-end application calculate the correct time for the user. This way you will be more independent of any changes.

Backup and restore MySQL

Backup and restore will be necessary when you are going to switch MySQL versions. Each version has its own data directory and is fully independent of other versions. In case you want to switch to a different version, but still want to have your MySQL databases present, you must first backup the databases of your current version and import them into the new version.

There are multiple ways to backup and restore. Choose the one which is most convenient for you.

Table of Contents

- *Backup*
 - *Mysqldump-secure*
 - * *List backups*
 - * **.info files*
 - *mysqldump*
 - *phpMyAdmin*
 - *Adminer*
- *Restore*
 - *mysql*
 - * **.sql file*
 - * **.sql.gz file*
 - * **.sql.tar.gz file*
 - *phpMyAdmin*
 - *Adminer*

13.1 Backup

There are many different options to backup your MySQL database including some for the command line and some for using the Web interface. The recommended and fastest method is to use `mysqldump-secure`, as it will also add info files (*.info) to each database recording checksums, dump date, dump options as well as the server version it came from.

13.1.1 Mysqldump-secure

`mysqldump-secure` is bundled, setup and ready to use in every PHP container. You can run it without any arguments and it will dump each available database as a separated compressed file. Backups will be located in `./backups/mysql/` inside the Devilbox git directory or in `/shared/backups/mysql/` inside the PHP container.

To have your backups in place is just three commands away:

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Enter the PHP container
host> ./shell.sh

# Run mysqldump-secure
devilbox@php-7.1.6 in /shared/httpd $ mysqldump-secure

[INFO] (OPT): Logging enabled
[INFO] (OPT): MySQL SSL connection disabled
[INFO] (OPT): Compression enabled
[INFO] (OPT): Encryption disabled
[INFO] (OPT): Deletion disabled
[INFO] (OPT): Nagios log disabled
[INFO] (OPT): Info files enabled
[INFO] (SQL): 1/3 Skipping: information_schema (DB is ignored)
[INFO] (SQL): 2/3 Dumping: mysql (0.66 MB) 1 sec (0.13 MB)
[INFO] (SQL): 3/3 Skipping: performance_schema (DB is ignored)
[OK] Finished successfully
```

List backups

Let's see where to find the backups inside the PHP container:

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Enter the PHP container
host> ./shell.sh

# Show directory output
devilbox@php-7.1.6 in /shared/httpd $ ls -l /shared/backups/mysql/

-rw-r--r-- 1 devilbox 136751 Jun 17 13:31 2017-06-17_13-31__mysql.sql.gz
-rw-r--r-- 1 devilbox 2269 Jun 17 13:31 2017-06-17_13-31__mysql.sql.gz.info
```

Let's do the same again and see where to find the backups in the Devilbox git directory

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Show directory output
host> ls -l backups/mysql/

-rw-r--r-- 1 cytopia 136751 Jun 17 13:31 2017-06-17_13-31__mysql.sql.gz
-rw-r--r-- 1 cytopia 2269 Jun 17 13:31 2017-06-17_13-31__mysql.sql.gz.info
```

*.info files

The *.info file will hold many useful information in case you need to debug any problems occurred during backups. Let's have a look at one of them:

```
host> cat ./backups/mysql/2017-06-17_13-31__mysql.sql.gz.info
```

Listing 1: 2017-06-17_13-31__mysql.sql.gz.info

```
; mysqldump-secure backup record
; Do not alter this file!
; Creation of this file can be turned off via config file.

; =====
; = Local system information
; =====
[mysqldump-secure]
version      = /usr/local/bin/mysqldump-secure (0.16.3)
vdate        = 2016-08-18
config       = /etc/mysqldump-secure.conf

[system]
uname        = Linux 4.4.0-79-generic
hostname     =
user         = devilbox
group        = devilbox

[tools]
mysqldump    = /usr/bin/mysqldump (10.14 Distrib 5.5.52-MariaDB) [for Linux (x86_64)]
mysql        = /usr/bin/mysql (15.1 Distrib 5.5.52-MariaDB) [for Linux (x86_64) using
↳ readline 5.1]
compressor   = /usr/bin/gzip (gzip 1.5)
encryptor    = Not used

; =====
; = Database / File information
; =====
[database]
db_name      = mysql
db_size      = 687326 Bytes (0.66 MB)
tbl_cnt      = 30

[file]
file_path    = /shared/backups/mysql
file_name    = 2017-06-17_13-31__mysql.sql.gz
file_size    = 136751 Bytes (0.13 MB)
```

(continues on next page)

(continued from previous page)

```

file_chmod = 0644
file_owner = devilbox
file_group = devilbox
file_mtime = 1497699116 (2017-06-17 13:31:56 CEST [+0200])
file_md5    = 8d1a6c38f81c691bc4b490e7024a4f72
file_sha    = 11fb85282ea866dfc69d29dc02a0418bebfea30e7e566c3c588a50987aceac2f

; =====
; = Dump procedure information
; =====
[mysqldump]
encrypted    = 0
compressed  = 1
arguments    = --opt --default-character-set=utf8 --events --triggers --routines --hex-
↳ blob --complete-insert --extended-insert --compress --lock-tables --skip-quick
duration     = 1 sec

[compression]
compressor   = gzip
arguments    = -9 --stdout

[encryption]
encryptor    =
algorithm    =
pubkey       =

; =====
; = Server information
; =====
[connection]
protocol     = mysql via TCP/IP
secured      = No SSL
arguments    = --defaults-file=/etc/mysqldump-secure.cnf

[server]
hostname     = 001b3750b549
port         = 3306
replica      = master
version      = MariaDB 10.1.23-MariaDB MariaDB Server

```

13.1.2 mysqldump

`mysqldump` is bundled with each PHP container and ready to use. To backup a database named `my_db_name` follow the below listed example which shows you how to do that from within the PHP container:

```

# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Enter the PHP container
host> ./shell.sh

# Start the backup
devilbox@php-7.1.6 in /shared/httpd $ mysqldump -h mysql -u root -p my_db_name > /
↳ shared/backups/mysql/my_db_name.sql

```

To find out more about the configuration and options of `mysqldump`, visit its project page under: <https://dev.mysql>.

com/doc/refman/5.7/en/mysqldump.html

13.1.3 phpMyAdmin

If you do not like to use the command line for backups, you can use [phpMyAdmin](#). It comes bundled with the devilbox intranet.

To find out more about the usage of phpMyAdmin, visit its project page under: <https://www.phpmyadmin.net>.

13.1.4 Adminer

If you do not like to use the command line for backups, you can use [Adminer](#). It comes bundled with the devilbox intranet.

To find out more about the usage of Adminer, visit its project page under: <https://www.adminer.org>.

13.2 Restore

13.2.1 mysql

In order to restore or import mysql databases on the command line, you need to use the `mysql` binary. Here are a few examples for different file types:

★ .sql file

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Enter the PHP container
host> ./shell.sh

# Start the import
devilbox@php-7.1.6 in /shared/httpd $ mysql -h mysql -u root -p my_db_name < /shared/
↳backups/mysql/my_db_name.sql
```

★ .sql.gz file

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Enter the PHP container
host> ./shell.sh

# Start the import
devilbox@php-7.1.6 in /shared/httpd $ zcat /shared/backups/mysql/my_db_name.sql.gz |
↳mysql -h mysql -u root -p my_db_name
```

***.sql.tar.gz file**

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Enter the PHP container
host> ./shell.sh

# Start the import
devilbox@php-7.1.6 in /shared/httpd $ tar xzOf /shared/backups/mysql/my_db_name.sql.
↳tar.gz | mysql -h mysql -u root -p my_db_name
```

13.2.2 phpMyAdmin

[phpMyAdmin](#) supports importing many different formats out-of-the-box. Simply select the compressed or uncompressed file and press **Go** in the import section of the web interface.

13.2.3 Adminer

[Adminer](#) supports importing of plain (*.sql) or gzipped compressed (*.sql.gz) files out-of-the-box. Simply select the compressed or uncompressed file and press **Execute** in the import section of the web interface.

CHAPTER 14

Backup and restore PostgreSQL

Backup and restore will be necessary when you are going to switch PostgreSQL versions. Each version has its own data directory and is fully independent of other versions. In case you want to switch to a different version, but still want to have your PostgreSQL databases present, you must first backup the databases of your current version and import them into the new version.

There are multiple ways to backup and restore. Choose the one which is most convenient for you.

Table of Contents

- *Backup*
 - *pg_dump*
 - *Adminer*
- *Restore*
 - *psql*
 - * **.sql file*
 - * **.sql.gz file*
 - * **.sql.tar.gz file*
 - *Adminer*

14.1 Backup

14.1.1 pg_dump

`pg_dump` is bundled with each PHP container and ready to use. To backup a database named `my_db_name` follow the below listed example:

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Enter the PHP container
host> ./shell.sh

# Run pg_dump
devilbox@php-7.1.6 in /shared/httpd $ pg_dump -h pgsql -U postgres -W my_db_name > /
↳ shared/backups/pgsql/my_db_name.sql
```

To find out more about the configuration and options of `pg_dump`, visit its project page under: <https://www.postgresql.org/docs/current/static/backup-dump.html>.

14.1.2 Adminer

If you do not like to use the command line for backups, you can use [Adminer](#). It comes bundled with the devilbox intranet.

To find out more about the usage of Adminer, visit its project page under: <https://www.adminer.org>.

14.2 Restore

14.2.1 psql

In order to restore or import PostgreSQL databases on the command line, you need to use `psql`. Here are a few examples for different file types:

★ .sql file

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Enter the PHP container
host> ./shell.sh

# Start the import
devilbox@php-7.1.6 in /shared/httpd $ psql -h pgsql -U postgres -W my_db_name < /
↳ shared/backups/pgsql/my_db_name.sql
```

★ .sql.gz file

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Enter the PHP container
host> ./shell.sh

# Start the import
devilbox@php-7.1.6 in /shared/httpd $ zcat /shared/backups/pgsql/my_db_name.sql.gz |
↳ psql -h pgsql -U postgres -W my_db_name
```


***.sql.tar.gz file**

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Enter the PHP container
host> ./shell.sh

# Start the import
devilbox@php-7.1.6 in /shared/httpd $ tar xzOf /shared/backups/pgsql/my_db_name.sql.
↳tar.gz | psql -h pgsql -U postgres -W my_db_name
```

14.2.2 Adminer

[Adminer](#) supports importing of plain (*.sql) or gzipped compressed (*.sql.gz) files out-of-the-box. Simply select the compressed or uncompressed file and press **Execute** in the import section of the web interface.

CHAPTER 15

Backup and restore MongoDB

Backup and restore will be necessary when you are going to switch MongoDB versions. Each version has its own data directory and is fully independent of other versions. In case you want to switch to a different version, but still want to have your MongoDB databases present, you must first backup the databases of your current version and import them into the new version.

There are multiple ways to backup and restore. Choose the one which is most convenient for you.

Table of Contents

- *Backup*
 - *mongodump*
- *Restore*
 - *mongorestore*

15.1 Backup

15.1.1 mongodump

`mongodump` is bundled with each PHP container and ready to use. To backup all databases follow the below listed example:

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Enter the PHP container
host> ./shell.sh
```

(continues on next page)

(continued from previous page)

```
# Run mongodump
devilbox@php-7.1.6 in /shared/httpd $ mongodump --out /shared/backups/mongo
```

To find out more about the configuration and options of mongodump, visit its project page under: <https://docs.mongodb.com/manual/reference/program/mongodump>.

15.2 Restore

15.2.1 mongorestore

mongorestore is bundled with each PHP container and ready to use. To restore all MongoDB databases follow the below listed example:

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Enter the PHP container
host> ./shell.sh

# Start the restore/import from /shared/backups/mongo
devilbox@php-7.1.6 in /shared/httpd $ mongorestore /shared/backups/mongo
```

To find out more about the configuration and options of mongorestore, visit its project page under: <https://docs.mongodb.com/manual/reference/program/mongorestore/>.

Communicating with external hosts

This tutorial shows you how to connect the Devilbox to running Docker container outside the Devilbox network, i.e. Docker container you have started separately.

Table of Contents

- *Prerequisites*
 - *Host IP: Docker on Linux*
 - *Host IP: Docker for Mac*
 - * *Docker 18.03.0-ce+ and Docker compose 1.20.1+*
 - * *Docker 17.12.0-ce+ and Docker compose 1.18.0+*
 - * *Docker 17.06.0-ce+ and Docker compose 1.14.0+*
 - *Host IP: Docker for Windows*
 - * *Docker 18.03.0-ce+ and Docker compose 1.20.1+*
 - * *Docker 17.06.0-ce+ and Docker compose 1.14.0+*
- *Make DNS available to the Devilbox*
 - *Adding extra hosts*
 - *Example*
 - * *Mapping on Linux*
 - * *Mapping on MacOS*
 - * *Mapping on Windows*
 - *Auto DNS*
- *Further reading*

16.1 Prerequisites

There are two things you need to make sure of are met beforehand:

1. The external Docker container must expose its ports on all interfaces on your host operating system
2. The IP by which the host is reachable from within the Devilbox container.

16.1.1 Host IP: Docker on Linux

If you run Docker on Linux the host IP is always `172.16.238.1`, which is the default gateway IP address within the Devilbox bridge network (see `docker-compose.yml`).

By default Docker on Linux does not have CNAME's of the host computer as for example with MacOS or Windows, therefore two custom CNAME's have been added by the Devilbox in order to emulate the same behaviour:

- CNAME: `docker.for.lin.host.internal`
- CNAME: `docker.for.lin.localhost`

16.1.2 Host IP: Docker for Mac

If you run Docker for Mac, an IP address is not necessary as it already provides a CNAME which will always point to the IP address of your host operating system. Depending on the Docker version this CNAME will differ:

Docker 18.03.0-ce+ and Docker compose 1.20.1+

CNAME: `host.docker.internal`

Docker 17.12.0-ce+ and Docker compose 1.18.0+

CNAME: `docker.for.mac.host.internal`

Docker 17.06.0-ce+ and Docker compose 1.14.0+

CNAME: `docker.for.mac.localhost`

16.1.3 Host IP: Docker for Windows

If you run Docker for Windows, an IP address is not necessary as it already provides a CNAME which will always point to the IP address of your host operating system. Depending on the Docker version this CNAME will differ:

Docker 18.03.0-ce+ and Docker compose 1.20.1+

CNAME: `docker.for.win.host.internal`

Docker 17.06.0-ce+ and Docker compose 1.14.0+

CNAME: `docker.for.win.host.localhost`

16.2 Make DNS available to the Devilbox

Inside each Devilbox Docker container you can already connect to all host ports (if they are bound to all interfaces) by the above specified IP addresses or CNAME's. You can however also create a custom DNS entry for convenience or if an external web server requires a special vhost name.

16.2.1 Adding extra hosts

Extra hosts (hostname and IP address mappings or hostname and CNAME mappings) can be set in the `.env` file.

See also:

[*EXTRA_HOSTS*](#)

16.2.2 Example

Let's assume another Docker container is running on your host, which must be accessed by the exact name of `mywebserver.loc` in order to respond by that virtual host name.

Mapping on Linux

If you are running Linux as your host operating system you would use the IP address of the host computer which was identified as `172.16.238.1`.

Listing 1: `.env`

```
EXTRA_HOSTS=mywebserver.loc=172.16.238.1
```

or

Listing 2: `.env`

```
EXTRA_HOSTS=mywebserver.loc=docker.for.lin.host.internal
```

or

Listing 3: `.env`

```
EXTRA_HOSTS=mywebserver.loc=docker.for.lin.localhost
```

Mapping on MacOS

If you are running MacOS as your host operating system you would use one of the identified CNAME's (depending on your Docker version).

Listing 4: `.env`

```
EXTRA_HOSTS=mywebserver.loc=host.docker.internal
```

The CNAME `host.docker.internal` will be resolved to an IP address during startup and `mywebserver.loc`'s DNS record will point to that IP address.

Mapping on Windows

If you are running Windows as your host operating system you would use one of the identified CNAME's (depending on your Docker version).

Listing 5: .env

```
EXTRA_HOSTS=mywebserver.loc=docker.for.win.host.internal
```

The CNAME `docker.for.win.host.internal` will be resolved to an IP address during startup and `mywebserver.loc`'s DNS record will point to that IP address.

16.2.3 Auto DNS

If you also turned on *Auto-DNS* these extra hosts will then also be available to your host operating system as well.

16.3 Further reading

See also:

- *EXTRA_HOSTS*
- *Auto-DNS*

Add your own Docker image

This section is all about customizing the Devilbox and its Docker images specifically to your needs.

Table of Contents

- *Prerequisites*
- *What information do you need?*
- *How to add a new service?*
 - *Generic example*
 - * *A single new service*
 - * *Two new services*
 - *CockroachDB example*
- *How to start the new service?*
- *Further reading*

17.1 Prerequisites

The new Docker image definition will be added to a file called `docker-compose.override.yml`. So before going any further, read the following section that shows you how to create this file for the Devilbox as well as what pitfalls to watch out for.

See also:

docker-compose.override.yml

17.2 What information do you need?

1. `<name>` - A name, which you can use to refer in the `docker-compose` command
2. `<image-name>` - The Docker image name itself
3. `<image-version>` - The Docker image tag
4. `<unused-ip-address>` - An unused IP address from the devilbox network (found inside `docker-compose.yml`)

17.3 How to add a new service?

17.3.1 Generic example

A single new service

Open `docker-compose.override.yml` with your favourite editor and paste the following snippets into it.

Listing 1: `docker-compose.override.yml`

```
version: '2.1'
services:
  # Your custom Docker image here:
  <name>:
    image: <image-name>:<image-version>
    networks:
      app_net:
        ipv4_address: <unused-ip-address>
  # For ease of use always automatically start these:
  depends_on:
    - bind
    - php
    - httpd
  # End of custom Docker image
```

Note:

- `<name>` has to be replaced with any name of your choice
 - `<image-name>` has to be replaced with the name of the Docker image
 - `<image-version>` has to be replaced with the tag of the Docker image
 - `<unused-ip-address>` has to be replaced with an unused IP address
-

Two new services

Listing 2: `docker-compose.override.yml`

```
version: '2.1'
services:
  # Your first custom Docker image here:
```

(continues on next page)

(continued from previous page)

```
<name1>:
  image: <image1-name>:<image1-version>
  networks:
    app_net:
      ipv4_address: <unused-ip-address1>
  # For ease of use always automatically start these:
  depends_on:
    - bind
    - php
    - httpd
  # End of first custom Docker image
  # Your second custom Docker image here:
<name2>:
  image: <image2-name>:<image2-version>
  networks:
    app_net:
      ipv4_address: <unused-ip-address2>
  # For ease of use always automatically start these:
  depends_on:
    - bind
    - php
    - httpd
  # End of second custom Docker image
```

Note:

- <name1> has to be replaced with any name of your choice
- <image1-name> has to be replaced with the name of the Docker image
- <image1-version> has to be replaced with the tag of the Docker image
- <unused-ip-address1> has to be replaced with an unused IP address

Note:

- <name2> has to be replaced with any name of your choice
- <image2-name> has to be replaced with the name of the Docker image
- <image2-version> has to be replaced with the tag of the Docker image
- <unused-ip-address2> has to be replaced with an unused IP address

17.3.2 CockroachDB example

Gather the requirements for the [Cockroach DB](#) Docker image:

1. Name: cockroach
2. Image: cockroachdb/cockroach
3. Tag: latest
4. IP: 172.16.238.200

Now add the information to `docker-compose.override.yml`:

Listing 3: docker-compose.override.yml

```
version: '2.1'
services:
  # Your custom Docker image here:
  cockroach:
    image: cockroachdb/cockroach:latest
    command: start --insecure
    networks:
      app_net:
        ipv4_address: 172.16.238.200
  # For ease of use always automatically start these:
  depends_on:
    - bind
    - php
    - httpd
  # End of custom Docker image
```

17.4 How to start the new service?

The following will bring up your service including all of its dependent services, as defined with `depends_on` (bind, php and httpd). You need to replace `<name>` with the name you have chosen.

```
host> docker-compose up <name>
```

In the example of Cockroach DB the command would look like this

```
host> docker-compose up cockroach
```

17.5 Further reading

See also:

- *docker-compose.override.yml*
- *Overwrite existing Docker image*

Overwrite existing Docker image

This section is all about customizing the Devilbox and its Docker images specifically to your needs.

Table of Contents

- *Prerequisites*
- *What information do you need?*
- *How to overwrite a service?*
 - *Generic steps*
 - *Overwrite Docker image for the bind service*
- *Further reading*

18.1 Prerequisites

The new Docker image overwrite will be added to a file called `docker-compose.override.yml`. So before going any further, read the following section that shows you how to create this file for the Devilbox as well as what pitfalls to watch out for.

See also:

docker-compose.override.yml

18.2 What information do you need?

1. The service to overwrite

18.3 How to overwrite a service?

18.3.1 Generic steps

1. Copy the whole service definition from `docker-compose.yml` to `docker-compose.override.yml`
2. Remove anything unnecessary
3. Adjust the values you need

18.3.2 Overwrite Docker image for the bind service

The following example is using the `bind` service and overrides the Docker image to illustrate how this is done :

First you simply copy the while definition of the `bind` service from `docker-compose.yml` to `docker-compose.override.yml`:

Listing 1: `docker-compose.override.yml`

```
version: '2.1'
services:
  bind:
    image: cytopia/bind:0.11
    restart: always
    ports:
      # [local-machine:]local-port:docker-port
      - "${LOCAL_LISTEN_ADDR}${HOST_PORT_BIND:-1053}:53"
      - "${LOCAL_LISTEN_ADDR}${HOST_PORT_BIND:-1053}:53/udp"

    environment:
      ##
      ## Debug?
      ##
      - DEBUG_ENTRYPOINT=${DEBUG_COMPOSE_ENTRYPOINT}
      - DOCKER_LOGS=1

      ##
      ## Bind settings
      ##
      - WILDCARD_ADDRESS=172.16.238.11
      - DNS_FORWARDER=${BIND_DNS_RESOLVER:-8.8.8.8,8.8.4.4}

    dns:
      - 127.0.0.1

    networks:
      app_net:
        ipv4_address: 172.16.238.100
```

The second step is to remove everything that you do not need to overwrite:

Listing 2: `docker-compose.override.yml`

```
version: '2.1'
services:
  bind:
    image: cytopia/bind:0.11
```

The last step is to actually adjust the value you want to change for the bind service:

Listing 3: docker-compose.override.yml

```
version: '2.1'
services:
  bind:
    image: someother/bind:latest
```

18.4 Further reading

See also:

- *docker-compose.override.yml*
- *Add your own Docker image*

Adding Sub domains

This tutorial gives you a brief overview how to serve your project under one subdomain via the project directory name as well as how to serve one project with multiple subdomains with a customized virtual host config via `vhost-gen`.

Table of Contents

- *Single sub domain for one project*
- *Multiple sub domains for one project*
 - *Prerequisite*
 - *Apache 2.2*
 - * *Adding `www` sub domain*
 - *Step 1: Add DNS entry*
 - *Step 2: Adjust `apache22.yml`*
 - *Step 3: Apply new changes*
 - * *Adding catch-all sub domain*
 - *Step 1: Add DNS entry*
 - *Step 2: Adjust `apache22.yml`*
 - *Step 3: Apply new changes*
 - *Apache 2.4*
 - *Nginx*
 - * *Adding `www` sub domain*
 - *Step 1: Add DNS entry*
 - *Step 2: Adjust `nginx.yml`*

- *Step 3: Apply new changes*
- * *Adding catch-all sub domain*
 - *Step 1: Add DNS entry*
 - *Step 2: Adjust nginx.yml*
 - *Step 3: Apply new changes*
- *Apply changes*
- *Checklist*

19.1 Single sub domain for one project

When you just want to serve your project under a sub domain, you simply name your project directory by the name of it. See the following examples how you build up your project URL.

| Project dir | TLD_SUFFIX | Project URL |
|----------------|------------|---------------------------|
| my-test | loc | http://my-test.loc |
| www.my-test | loc | http://www.my-test.loc |
| t1.www.my-test | loc | http://t1.www.my-test.loc |
| my-test | com | http://my-test.com |
| www.my-test | com | http://www.my-test.com |
| t2.www.my-test | com | http://t2.www.my-test.com |

Whatever name you want to have in front of the TLD_SUFFIX is actually just the directory you create. Generically, it looks like this:

| Project dir | TLD_SUFFIX | Project URL |
|-------------|------------|-------------------------|
| <dir-name> | <tld> | http://<dir-name>.<tld> |

19.2 Multiple sub domains for one project

When you want to have multiple domains and/or sub domains for one project (such as in the case of Wordpress multi-sites), you will need to customize your virtual host config for this project and make the web server allow to serve your files by different server names.

Each web server virtual host is auto-generated by a tool called **vhost-gen**. **vhost-gen** allows you to overwrite its default generation process via templates. Those templates can be added to each project, giving you the option to customize the virtual host of this specific project.

Note:

Customized virtual host (vhost-gen) Ensure you have read and understand how to customize your virtual host with **vhost-gen**.

HTTPD_TEMPLATE_DIR Ensure you know what this variable does inside your **.env** file.

Important: When adjusting **vhost-gen** templates for a project you have to do **one** of the following:

- Restart the devilbox
- Or rename your project directory to some other name and then rename it back to its original name.

More information here: [Apply Changes](#)

Warning: Pay close attention that you do not use TAB (`\t`) characters for indenting the vhost-gen yaml files. Some editors might automatically indent using TABs, so ensure they are replaced with spaces. If TAB characters are present, those files become invalid and won't work. <https://github.com/cytopia/devilbox/issues/142>

You can use the bundled `yamllint` binary inside the container to validate your config.

See also:

- [Work inside the container](#)
- [Available tools](#)

19.2.1 Prerequisite

Let's assume the following settings.

| Variable | Value |
|--------------------------------------|-----------------------------|
| Devilbox path | /home/user/devilbox |
| <code>HTTPD_TEMPLATE_DIR</code> | .devilbox |
| <code>HOST_PATH_HTTPD_DATADIR</code> | ./data/www |
| <code>TLD_SUFFIX</code> | loc |
| Project name/directory | project-1 (Ensure it exist) |

Ensure that the default vhost-gen templates have been copied to your projects template directory:

```
# Navigate to the Devilbox directory
host> cd ./home/user/devilbox

# Create template directory in your project
host> mkdir ./data/www/project-1/.devilbox

# Copy vhost-gen templates
host> cp templates/vhost-gen/* ./data/www/project-1/.devilbox
```

By having done all prerequisite, your project should be available under <http://my-project-1.loc>

Now you are all set and we can dive into the actual configuration.

19.2.2 Apache 2.2

Adding `www` sub domain

Let's also make this project available under <http://www.my-project-1.loc>

Step 1: Add DNS entry

The first step would be to add an additional DNS entry for `www.my-project-1.loc`. See here how to do that for Linux, MacOS or Windows: *Step 4: create a DNS entry*

DNS is in place, however when you visit <http://www.my-project-1.loc>, you will end up seeing the Devilbox intranet, because this is the default host when no match has been found.

Step 2: Adjust apache22.yml

Next you will have to adjust the Apache 2.2 vhost configuration template. The current default template looks similar to the one shown below (**Note:** Only the `vhost` : sub section is shown here).

Listing 1: `/home/user/devilbox/data/www/project-1/devilbox/apache22.yml`

```
vhost: |
  <VirtualHost __DEFAULT_VHOST__:__PORT__>
    ServerName __VHOST_NAME__

    CustomLog "__ACCESS_LOG__" combined
    ErrorLog "__ERROR_LOG__"

    __VHOST_DOCROOT__
    __VHOST_RPROXY__
    __PHP_FPM__
    __ALIASES__
    __DENIES__
    __SERVER_STATUS__
    # Custom directives
    __CUSTOM__
  </VirtualHost>
```

All you will have to do, is to add another `ServerName` directive:

Listing 2: `/home/user/devilbox/data/www/project-1/devilbox/apache22.yml`

```
vhost: |
  <VirtualHost __DEFAULT_VHOST__:__PORT__>
    ServerName __VHOST_NAME__
    ServerName www.__VHOST_NAME__

    CustomLog "__ACCESS_LOG__" combined
    ErrorLog "__ERROR_LOG__"

    __VHOST_DOCROOT__
    __VHOST_RPROXY__
    __PHP_FPM__
    __ALIASES__
    __DENIES__
    __SERVER_STATUS__
    # Custom directives
    __CUSTOM__
  </VirtualHost>
```

Step 3: Apply new changes

The **last step** is to actually to apply those changes. This is equal for all web servers. Go to [Apply changes](#) and follow the steps.

Afterwards you can go and visit <http://www.my-project-1.loc> and you should see the same page as if you visit <http://my-project-1.loc>

Adding catch-all sub domain

Let's also make this project available under any sub domain.

Step 1: Add DNS entry

The first step would be to add DNS entries for all sub domains you require. See here how to do that for Linux, MacOS or Windows: [Step 4: create a DNS entry](#)

This however is not really convenient. If you have basically infinite sub domains (via catch-all), you should consider using Auto-DNS instead: [Auto-DNS](#).

Step 2: Adjust apache22.yml

Next you will have to adjust the Apache 2.2 vhost configuration template. The current default template looks similar to the one shown below (**Note:** Only the `vhost :` sub section is shown here).

Listing 3: /home/user/devilbox/data/www/project-1/devilbox/apache22.yml

```
vhost: |
  <VirtualHost __DEFAULT_VHOST__:__PORT__>
    ServerName __VHOST_NAME__

    CustomLog "__ACCESS_LOG__" combined
    ErrorLog "__ERROR_LOG__"

    __VHOST_DOCROOT__
    __VHOST_RPROXY__
    __PHP_FPM__
    __ALIASES__
    __DENIES__
    __SERVER_STATUS__
    # Custom directives
    __CUSTOM__
  </VirtualHost>
```

All you will have to do, is to add another `ServerName` directive which does catch-all:

Listing 4: /home/user/devilbox/data/www/project-1/devilbox/apache22.yml

```
vhost: |
  <VirtualHost __DEFAULT_VHOST__:__PORT__>
    ServerName __VHOST_NAME__
    ServerName *.__VHOST_NAME__
```

(continues on next page)

(continued from previous page)

```
CustomLog    "__ACCESS_LOG__" combined
ErrorLog     "__ERROR_LOG__"

__VHOST_DOCROOT__
__VHOST_RPROXY__
__PHP_FPM__
__ALIASES__
__DENIES__
__SERVER_STATUS__
    # Custom directives
__CUSTOM__
</VirtualHost>
```

Step 3: Apply new changes

The **last step** is to actually to apply those changes. This is equal for all web servers. Go to [Apply changes](#) and follow the steps.

19.2.3 Apache 2.4

The procedure for Apache 2.4 is exactly the same as for Apache 2.2, even the syntax is identical. The only difference is that you need to adjust `apache24.yml` instead of `apache22.yml`.

Just go up one section: [Apache 2.2](#)

19.2.4 Nginx

The procedure for Nginx is also exactly the same as for Apache 2.4, however the syntax of its `nginx.yml` file is slightly different, that's why the whole tutorial will be repeated here fitted for Nginx.

Adding `www` sub domain

Let's also make this project available under `http://www.my-project-1.loc`

Step 1: Add DNS entry

The first step would be to add an additional DNS entry for `www.my-project-1.loc`. See here how to do that for Linux, MacOS or Windows: [Step 4: create a DNS entry](#)

DNS is in place, however when you visit <http://www.my-project-1.loc>, you will end up seeing the Devilbox intranet, because this is the default host when no match has been found.

Step 2: Adjust `nginx.yml`

Next you will have to adjust the Nginx vhost configuration template. The current default template looks similar to the one shown below (**Note:** Only the `vhost :` sub section is shown here).

Listing 5: /home/user/devilbox/data/www/project-1/devilbox/nginx.yml

```
vhost: |
  server {
    listen      __PORT__DEFAULT_VHOST__;
    server_name __VHOST_NAME__;

    access_log  "__ACCESS_LOG__" combined;
    error_log   "__ERROR_LOG__" warn;

    __VHOST_DOCROOT__
    __VHOST_RPROXY__
    __PHP_FPM__
    __ALIASES__
    __DENIES__
    __SERVER_STATUS__
    # Custom directives
    __CUSTOM__
  }
```

All you will have to do, is to extend the `server_name` directive:

Listing 6: /home/user/devilbox/data/www/project-1/devilbox/nginx.yml

```
vhost: |
  server {
    listen      __PORT__DEFAULT_VHOST__;
    server_name __VHOST_NAME__ www.__VHOST_NAME__;

    access_log  "__ACCESS_LOG__" combined;
    error_log   "__ERROR_LOG__" warn;

    __VHOST_DOCROOT__
    __VHOST_RPROXY__
    __PHP_FPM__
    __ALIASES__
    __DENIES__
    __SERVER_STATUS__
    # Custom directives
    __CUSTOM__
  }
```

Step 3: Apply new changes

The **last step** is to actually to apply those changes. This is equal for all web servers. Go to [Apply changes](#) and follow the steps.

Afterwards you can go and visit <http://www.my-project-1.loc> and you should see the same page as if you visit <http://my-project-1.loc>

Adding catch-all sub domain

Let's also make this project available under any sub domain.

Step 1: Add DNS entry

The first step would be to add DNS entries for all sub domains you require. See here how to do that for Linux, MacOS or Windows: [Step 4: create a DNS entry](#)

This however is not really convenient. If you have basically infinite sub domains (via catch-all), you should consider using Auto-DNS instead: [Auto-DNS](#).

Step 2: Adjust nginx.yml

Next you will have to adjust the Nginx vhost configuration template. The current default template looks similar to the one shown below (**Note:** Only the `vhost :` sub section is shown here).

Listing 7: `/home/user/devilbox/data/www/project-1/.devilbox/nginx.yml`

```
vhost: |
  server {
    listen      __PORT__ DEFAULT_VHOST__;
    server_name  __VHOST_NAME__;

    access_log   "__ACCESS_LOG__" combined;
    error_log    "__ERROR_LOG__" warn;

    __VHOST_DOCROOT__
    __VHOST_RPROXY__
    __PHP_FPM__
    __ALIASES__
    __DENIES__
    __SERVER_STATUS__
    # Custom directives
    __CUSTOM__
  }
```

All you will have to do, is to extend the `server_name` directive with a catch-all:

Listing 8: `/home/user/devilbox/data/www/project-1/.devilbox/nginx.yml`

```
vhost: |
  server {
    listen      __PORT__ DEFAULT_VHOST__;
    server_name  __VHOST_NAME__ *.__VHOST_NAME__;

    access_log   "__ACCESS_LOG__" combined;
    error_log    "__ERROR_LOG__" warn;

    __VHOST_DOCROOT__
    __VHOST_RPROXY__
    __PHP_FPM__
    __ALIASES__
    __DENIES__
    __SERVER_STATUS__
    # Custom directives
    __CUSTOM__
  }
```


Step 3: Apply new changes

The **last step** is to actually to apply those changes. This is equal for all web servers. Go to [Apply changes](#) and follow the steps.

19.2.5 Apply changes

After having edited your vhost-gen template files, you still need to apply these changes. This can be achieved in two ways:

1. Restart the Devilbox
2. Rename your project directory back and forth

Let's cover the second step

```
# Navigate to the data directory
host> /home/user/devilbox/data/www

# Rename your project to something else
host> mv project-1 project-1.tmp

# Rename your project to its original name
host> mv project-1.tmp project-1
```

If you want to understand what is going on right now, check the docker logs for the web server.

```
# Navigate to the devilbox directory
host> /home/user/devilbox

# Check docker logs
host> docker-compose logs httpd

httpd_1 | vhostgen: [2018-03-18 11:46:52] Adding: project-1.tmp.loc
httpd_1 | watcherd: [2018-03-18 11:46:52] [OK] ADD: succeeded: /shared/httpd/
↪project-1.tmp
httpd_1 | watcherd: [2018-03-18 11:46:52] [OK] DEL: succeeded: /shared/httpd/
↪project-1
httpd_1 | watcherd: [2018-03-18 11:46:52] [OK] TRIGGER succeeded: /usr/local/
↪apache2/bin/httpd -k restart

httpd_1 | vhostgen: [2018-03-18 11:46:52] Adding: project-1loc
httpd_1 | watcherd: [2018-03-18 11:46:52] [OK] ADD: succeeded: /shared/httpd/
↪project-1
httpd_1 | watcherd: [2018-03-18 11:46:52] [OK] DEL: succeeded: /shared/httpd/
↪project-1.tmp
httpd_1 | watcherd: [2018-03-18 11:46:52] [OK] TRIGGER succeeded: /usr/local/
↪apache2/bin/httpd -k restart
```

What happened?

The directory changes have been noticed and a new virtual host has been created. This time however your new vhost-gen template has been read and the changes have applied.

19.2.6 Checklist

1. Template files are copied from `templates/vhost-gen/*` to your project template dir (as specified in `.env` via `HTTPD_TEMPLATE_DIR`)
2. Ensure the vhost-gen yaml files are valid (No tab characters)
3. When templates are edited, the Devilbox is either restarted or the project directory is renamed to something else and then renamed back to its original name

Change container versions

One of the core concepts of the Devilbox is to easily change between different versions of a specific service.

Table of Contents

- *Change PHP version*
 - *Stop the Devilbox*
 - *Edit the .env file*
 - *Start the Devilbox*
 - *Gotchas*
- *Change whatever version*
- *Checklist*

20.1 Change PHP version

20.1.1 Stop the Devilbox

Shut down the Devilbox in case it is still running:

```
# Navigate to the Devilbox directory
host> /home/user/devilbox

# Stop all container
host> docker-compose stop
```

20.1.2 Edit the `.env` file

Open the `.env` file with your favourite editor and navigate to the `PHP_SERVER` section. It will look something like this:

Listing 1: `.env`

```
#PHP_SERVER=5.4
#PHP_SERVER=5.5
#PHP_SERVER=5.6
#PHP_SERVER=7.0
PHP_SERVER=7.1
#PHP_SERVER=7.1
```

As you can see, all available values are already there, but commented. Only one is uncommented. In this example it is `7.1`, which is the PHP version that will be started, once the Devilbox starts.

To change this, simply uncomment your version of choice and save this file. Do not forget to comment (disable) any other version.

In order to enable PHP 5.5, you would change the `.env` file like this:

Listing 2: `.env`

```
#PHP_SERVER=5.4
PHP_SERVER=5.5
#PHP_SERVER=5.6
#PHP_SERVER=7.0
#PHP_SERVER=7.1
#PHP_SERVER=7.1
```

20.1.3 Start the Devilbox

Now save the file and you can start the Devilbox again.

```
# Navigate to the Devilbox directory
host> /home/user/devilbox

# Stop all container
host> docker-compose up php httpd bind
```

See also:

Start the Devilbox

20.1.4 Gotchas

If two versions are uncommented, always the last one takes precedence.

Consider this `.env` file:

Listing 3: `.env`

```
#PHP_SERVER=5.4
PHP_SERVER=5.5
#PHP_SERVER=5.6
```

(continues on next page)

(continued from previous page)

```
PHP_SERVER=7.0
#PHP_SERVER=7.1
#PHP_SERVER=7.1
```

Both, PHP 5.4 and PHP 7.0 are uncommented, however, when you start the Devilbox, it will use PHP 7.0 as this value overwrites any previous ones.

20.2 Change whatever version

When you have read how to change the PHP version, it should be fairly simple to change any service version. It behaves in the exact same way.

The variable names of all available services with changable versions are in the following format: `<SERVICE>_SERVER`. Just look through the *.env file*.

See also:

The following variables control service versions: *PHP_SERVER*, *HTTPD_SERVER*, *MYSQL_SERVER*, *PGSQL_SERVER*, *REDIS_SERVER*, *MEMCD_SERVER*, *MONGO_SERVER*

20.3 Checklist

1. Stop the Devilbox
2. Uncomment version of choice in *.env*
3. Start the Devilbox

Work inside the container

The Devilbox allows you to completely work inside the PHP container (no matter what version), instead of your host operating system.

This brings a lot of advantages, such as that you don't have to install any development tool on your OS or if you are on Windows, you get a full blown Linux environment.

Additionally, special port-bindings and forwards are in place that allows you to even interchangeably work locally or inside the container without having to alter any php config for database and other connections.

See also:

Available tools

Table of Contents

- *Enter the container*
 - *Entering from Linux or MacOS: `shell.sh`*
 - *Entering from Windows: `shell.bat`*
- *Inside the container*
 - *`devilbox user`*
 - *`root user`*
- *Leave the container*
- *Host to Container mappings*
 - *File and directory Permissions*
 - *Directory mappings*
 - *IP address mappings*
 - *Port mappings*

- *DNS mappings*
- *Checklist*

21.1 Enter the container

Entering the computer is fairly simple. The Devilbox ships with two scripts to do that. One for Linux and MacOS (`shell.sh`) and another one for Windows (`shell.bat`).

21.1.1 Entering from Linux or MacOS: `shell.sh`

```
# Navigate to the Devilbox directory
host> cd /path/to/devilbox

# Run provided script
host> ./shell.sh

# Now you are inside the PHP Linux container
devilbox@php-7.0.19 in /shared/httpd $
```

21.1.2 Entering from Windows: `shell.bat`

```
# Navigate to the Devilbox directory
C:/> cd C:/Users/user1/devilbox

# Run provided script
C:/Users/user1/devilbox> shell.bat

# Now you are inside the PHP Linux container
devilbox@php-7.0.19 in /shared/httpd $
```

21.2 Inside the container

21.2.1 `devilbox` user

By using the provided scripts to enter the container you will become the user `devilbox`. This user will have the same uid and gid as the user from your host operating system.

So no matter what files or directories you create inside the container, they will have the same permissions and uid/gid set your host operating system. This of course also works the other way round.

The uid and gid mappings are controlled via two `.env` variables called `NEW_UID` and `NEW_GID`

See also:

If you want to find out more about synchronized container permissions read up here: [*Synchronize container permissions*](#)

21.2.2 root user

Sometimes however it is also necessary to do some actions that require super user privileges. You can always become root inside the container by either impersonating it or by using `sudo` to issue commands.

By default `sudo` is configured to be used without passwords, so you can simply do the following:

```
# As user devilbox inside the container
devilbox@php-7.0.19 in /shared/httpd $ sudo su -

# You are now the root user
root@php-7.0.19 in /shared/httpd $
```

You can also use `sudo` to run commands with root privileges without having to become root first.

```
# As user devilbox inside the container
devilbox@php-7.0.19 in /shared/httpd $ sudo apt update
devilbox@php-7.0.19 in /shared/httpd $ sudo apt install nmap
```

21.3 Leave the container

When you are inside the container and want to return to your host operating, just type `exit` and you are out.

```
# As user devilbox inside the container
devilbox@php-7.0.19 in /shared/httpd $ exit

# You are now back on your host operating system
host>
```

21.4 Host to Container mappings

This section will give you an idea that there is actually no difference from inside the container and on your host operating system. Directory permissions, IP addresses, ports and DNS entries are fully synchronized allowing you to switch between container and host without having to change any settings.

21.4.1 File and directory Permissions

The username inside the container (`devilbox`) might be different from your local host operating system username, however its actual uid and gid will match. This is to ensure file and directory permissions are synchronized inside and outside the container and no matter from which side you create files and directories, it will always look as if they are owned by your system user.

The uid and gid mappings are controlled via two `.env` variables called `NEW_UID` and `NEW_GID`

21.4.2 Directory mappings

One thing you should understand is the relation between the directories on your host operating system and the corresponding directory inside the PHP container.

The location of the data directory (*HOST_PATH_HTTPD_DATADIR*) on your host computer is controlled via the `HOST_PATH_HTTPD_DATADIR` variable inside the `.env` file. No matter what location you set it to, inside the container it will always be mapped to `/shared/httpd`.

See the following table for a few examples:

| | Host operating system | Inside PHP container |
|----------|------------------------------|----------------------------|
| Data dir | <code>./www/data</code> | <code>/shared/httpd</code> |
| Data dir | <code>/home/user1/www</code> | <code>/shared/httpd</code> |
| Data dir | <code>/var/www</code> | <code>/shared/httpd</code> |

21.4.3 IP address mappings

The following table shows a mapping of IP addresses of available service from the perspective of your host operating system and from within the PHP container.

| Service | IP from host os | IP from within PHP container |
|--------------|-----------------|------------------------------|
| PHP | 127.0.0.1 | 127.0.0.1 |
| Apache/Nginx | 127.0.0.1 | 127.0.0.1 |
| MySQL | 127.0.0.1 | 127.0.0.1 |
| PostgreSQL | 127.0.0.1 | 127.0.0.1 |
| Redis | 127.0.0.1 | 127.0.0.1 |
| Memcached | 127.0.0.1 | 127.0.0.1 |
| MongoDB | 127.0.0.1 | 127.0.0.1 |

As you can see, everything is available under `127.0.0.1`.

The PHP container is using `socat` to forward the services from all other available containers to its own `127.0.0.1` address.

An example to access the MySQL database from either host or within the PHP container is the same:

```
# Access MySQL from your host operating system
host> mysql -h 127.0.0.1

# Access MySQL from within the PHP container
devilbox@php-7.0.19 in /shared/httpd $ mysql -h 127.0.0.1
```

Important: Do not use `localhost` to access the services, it does not map to `127.0.0.1` on all cases.

So when setting up a configuration file from your PHP project you would for example use `127.0.0.` as the host for your MySQL database connection:

```
<?php
// MySQL server connection
mysql_host = '127.0.0.1';
mysql_port = '3306';
mysql_user = 'someusername';
mysql_pass = 'somepassword';
?>
```

Imagine your PHP framework ships a command line tool to run database migration. You could run it from your host operating system or from within the PHP container. It would work from both sides as the connection to the database is exactly the same locally or within the container.

You could also even switch between the Devilbox and a locally installed LAMP stack and still use the same configuration.

Warning: The mapping of `127.0.0.1` to your host operating system does not work with *Docker Toolbox*.

21.4.4 Port mappings

By default, ports are also synchronized between host operating system (the ports that are exposed) and the ports within the PHP container. This is however also configurable inside the `.env` file.

| Service | Port from host os | Port from within PHP container |
|--------------|-------------------|--------------------------------|
| PHP | NA | 9000 |
| Apache/Nginx | 80 | 80 |
| MySQL | 3306 | 3306 |
| PostgreSQL | 5432 | 5432 |
| Redis | 6379 | 6379 |
| Memcached | 11211 | 11211 |
| MongoDB | 27017 | 27017 |

21.4.5 DNS mappings

All project DNS records are also available from inside the PHP container independent of the value of `TLD_SUFFIX`.

The PHP container is hooked up by default to the bundled DNS server and makes use *Auto-DNS*.

See also:

You can achieve the same on your host operating system by explicitly enabling `auto-dns`. See also: *Auto-DNS*.

21.5 Checklist

1. You know how to enter the PHP container
2. You know how to become root inside the PHP container
3. You know how to leave the container
4. You know that file and directory permissions are synchronized
5. You know that `127.0.0.1` is available on your host and inside the PHP container
6. You know that ports are the same inside the container and on your host os
7. You know that project urls are available inside the container and on your host
8. You know about the limitations of *Docker Toolbox*

This tutorial shows you how to enable and use Xdebug with the Devilbox.

See also:

If you are unsure of how to add custom `*.ini` files to the Devilbox, have a look at this section first: [php.ini](#)

Table of Contents

- *Enable Xdebug*
 - *Required for all OS*
 - * *default_enable*
 - * *remote_enable*
 - * *remote_handler*
 - * *remote_port*
 - * *remote_autostart*
 - * *idekey*
 - * *remote_log*
 - *Linux*
 - *MacOS (Docker for Mac)*
 - *MacOS (Docker Toolbox)*
 - *Windows (Docker for Windows)*
 - *Windows (Docker Toolbox)*
- *Configure your IDE*
 - *Required for all IDE*

- * *Path mapping*
- * *IDE key*
- * *Port*
- *Atom*
 - * *Linux*
 - * *MacOS (Docker for Mac)*
 - * *MacOS (Docker Toolbox)*
 - * *Windows (Docker for Windows)*
 - * *Windows (Docker Toolbox)*
- *PHPStorm*
 - * *Linux*
 - * *MacOS (Docker for Mac)*
 - * *MacOS (Docker Toolbox)*
 - * *Windows (Docker for Windows)*
 - * *Windows (Docker Toolbox)*
- *Sublime Text 3*
 - * *Linux*
 - * *MacOS (Docker for Mac)*
 - * *MacOS (Docker Toolbox)*
 - * *Windows (Docker for Windows)*
 - * *Windows (Docker Toolbox)*
- *Visual Studio Code*
 - * *Linux*
 - * *MacOS (Docker for Mac)*
 - * *MacOS (Docker Toolbox)*
 - * *Windows (Docker for Windows)*
 - * *Windows (Docker Toolbox)*

22.1 Enable Xdebug

This section shows you the minimum required `*.ini` settings to get xdebug to work with the Devilbox. It will also highlight the differences between operating system and Docker versions.

See also:

See here for how to add the `*.ini` values to the Devilbox: [php.ini](#).

Once you have configured Xdebug, you can verify it at the Devilbox intranet: http://localhost/info_php.php

22.1.1 Required for all OS

Additionally to the specific configurations for each operating system and Docker version you will probably also want to add the following to your ini file:

Listing 1: xdebug.ini

```
xdebug.default_enable=1
xdebug.remote_enable=1
xdebug.remote_handler=dbgp
xdebug.remote_port=9000
xdebug.remote_autostart=1
xdebug.idekey="PHPSTORM"
xdebug.remote_log=/var/log/php/xdebug.log
```

See also:

https://xdebug.org/docs/all_settings

default_enable

By enabling this, stacktraces will be shown by default on an error event. It is advisable to leave this setting set to 1.

remote_enable

This switch controls whether Xdebug should try to contact a debug client which is listening on the host and port as set with the settings `xdebug.remote_host` and `xdebug.remote_port`. If a connection can not be established the script will just continue as if this setting was 0.

remote_handler

Can be either `'php3'` which selects the old PHP 3 style debugger output, `'gdb'` which enables the GDB like debugger interface or `'dbgp'` - the debugger protocol. The DBGp protocol is the only supported protocol.

Note: Xdebug 2.1 and later only support `'dbgp'` as protocol.

remote_port

The port to which Xdebug tries to connect on the remote host. Port 9000 is the default for both the client and the bundled debugclient. As many clients use this port number, it is best to leave this setting unchanged.

remote_autostart

Normally you need to use a specific HTTP GET/POST variable to start remote debugging (see [Remote Debugging](#)). When this setting is set to 1, Xdebug will always attempt to start a remote debugging session and try to connect to a client, even if the GET/POST/COOKIE variable was not present.

idekey

Controls which IDE Key Xdebug should pass on to the DBGp debugger handler. The default is based on environment settings. First the environment setting `DBGP_IDEKEY` is consulted, then `USER` and as last `USERNAME`. The default

is set to the first environment variable that is found. If none could be found the setting has as default `''`. If this setting is set, it always overrides the environment variables.

For the sake of this tutorial we are going to use `PHPSTORM` as an example value.

remote_log

Keep the exact path of `/var/log/php/xdebug.log`. You will then have the log file available in the Devilbox log directory of the PHP version for which you have configured Xdebug.

Important: You can set the value of `xdebug.idekey` to whatever you like, however it is important to remember what value you have set. Throughout the examples in this tutorial it is assumed, that the value is `PHPSTORM`.

22.1.2 Linux

Listing 2: xdebug.ini

```
xdebug.remote_connect_back=1
```

22.1.3 MacOS (Docker for Mac)

Docker 18.03.0-ce+ and Docker compose 1.20.1+

Listing 3: xdebug.ini

```
xdebug.remote_host=host.docker.internal
xdebug.remote_connect_back=0
```

Docker 17.12.0-ce+ and Docker compose 1.18.0+

Listing 4: xdebug.ini

```
xdebug.remote_host=docker.for.mac.host.internal
xdebug.remote_connect_back=0
```

Docker 17.06.0-ce+ and Docker compose 1.14.0+

Listing 5: xdebug.ini

```
xdebug.remote_host=docker.for.mac.localhost
xdebug.remote_connect_back=0
```

If you have older versions, upgrade.

22.1.4 MacOS (Docker Toolbox)

Warning: This is a legacy solution, upgrade to Docker for Mac <https://docs.docker.com/toolbox>

22.1.5 Windows (Docker for Windows)

Docker 18.03.0-ce+ and Docker compose 1.20.1+

Listing 6: xdebug.ini

```
xdebug.remote_host=docker.for.win.host.internal
xdebug.remote_connect_back=0
```

Docker 17.06.0-ce+ and Docker compose 1.14.0+

Listing 7: xdebug.ini

```
xdebug.remote_host=docker.for.win.host.localhost
xdebug.remote_connect_back=0
```

If you have older versions, upgrade.

22.1.6 Windows (Docker Toolbox)

Warning: This is a legacy solution, upgrade to Docker for Windows <https://docs.docker.com/toolbox>

22.2 Configure your IDE

22.2.1 Required for all IDE

Path mapping

The path mapping is a mapping between the file path on your host operating system and the one inside the PHP Docker container.

The path on your host operating system is the one you have set in `HOST_PATH_HTTPD_DATADIR`. In case you have set a relative path in `.env`, ensure to retrieve the absolute path of it when setting up your IDE config.

The path inside the PHP Docker container is always `/shared/httpd`.

Important: Even though your path in `.env` for `HOST_PATH_HTTPD_DATADIR` might be relative (e.g. `./data/www`), you need to get the actual absolute path of it, when setting up your IDE.

IDE key

This is the value you have set in `xdebug.ini` for `xdebug.idekey`. In the example of this tutorial, the value was set to `PHPSTORM`.

Port

This is the value you have set in `xdebug.ini` for `xdebug.remote_port`. In the example of this tutorial, the value was set to `9000`.

22.2.2 Atom

1. Install `php-debug`
2. Configure `config.cson` (File -> Config...)
3. Adjust your `xdebug.ini`

For Atom, you need to provide a different `xdebug.idekey` in your `php.ini` file `xdebug.ini`:

Listing 8: `xdebug.ini`

```
xdebug.idekey=xdebug.atom
```

Linux

Listing 9: `launch.json`

```
"php-debug":  
{  
  ServerPort: 9000  
  PathMaps: [  
    "remotepath;localpath"  
    "/shared/httpd;/home/cytopia/repo/devilbox/data/www"  
  ]  
}
```

MacOS (Docker for Mac)

Todo: Help needed. Please provide your config.

MacOS (Docker Toolbox)

Todo: Help needed. Please provide your config.

Windows (Docker for Windows)

Todo: Help needed. Please provide your config.

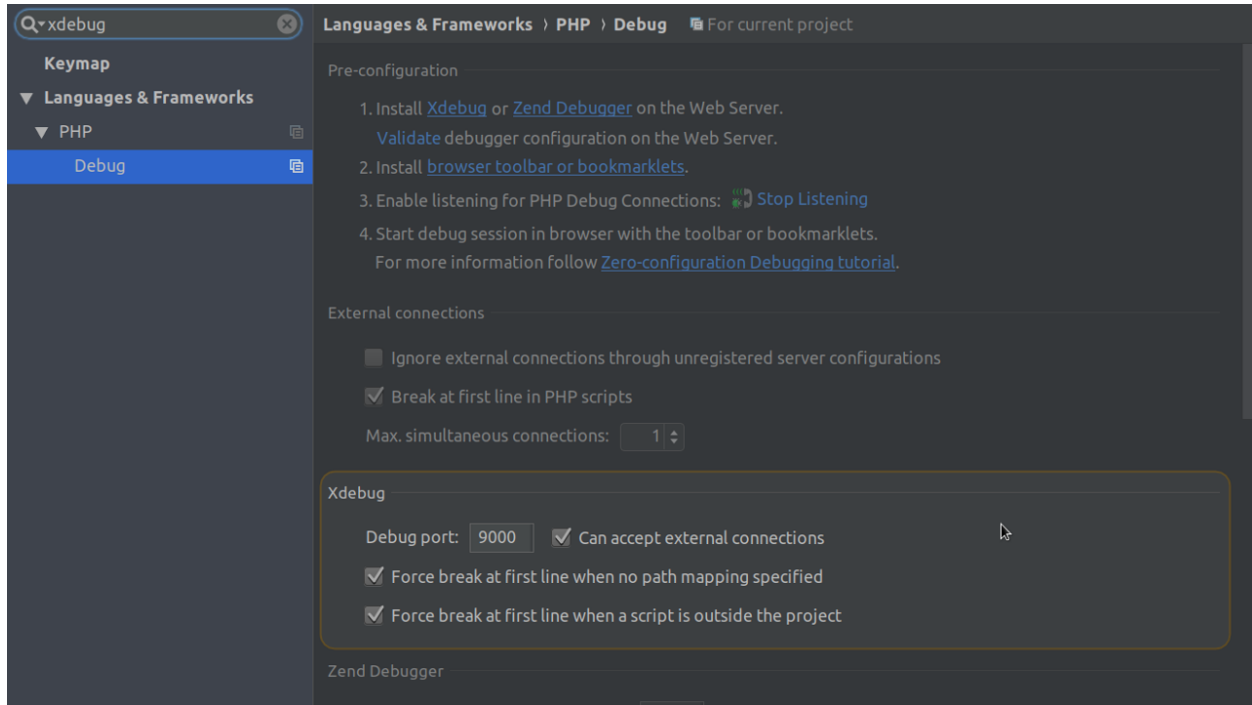
Windows (Docker Toolbox)

Todo: Help needed. Please provide your config.

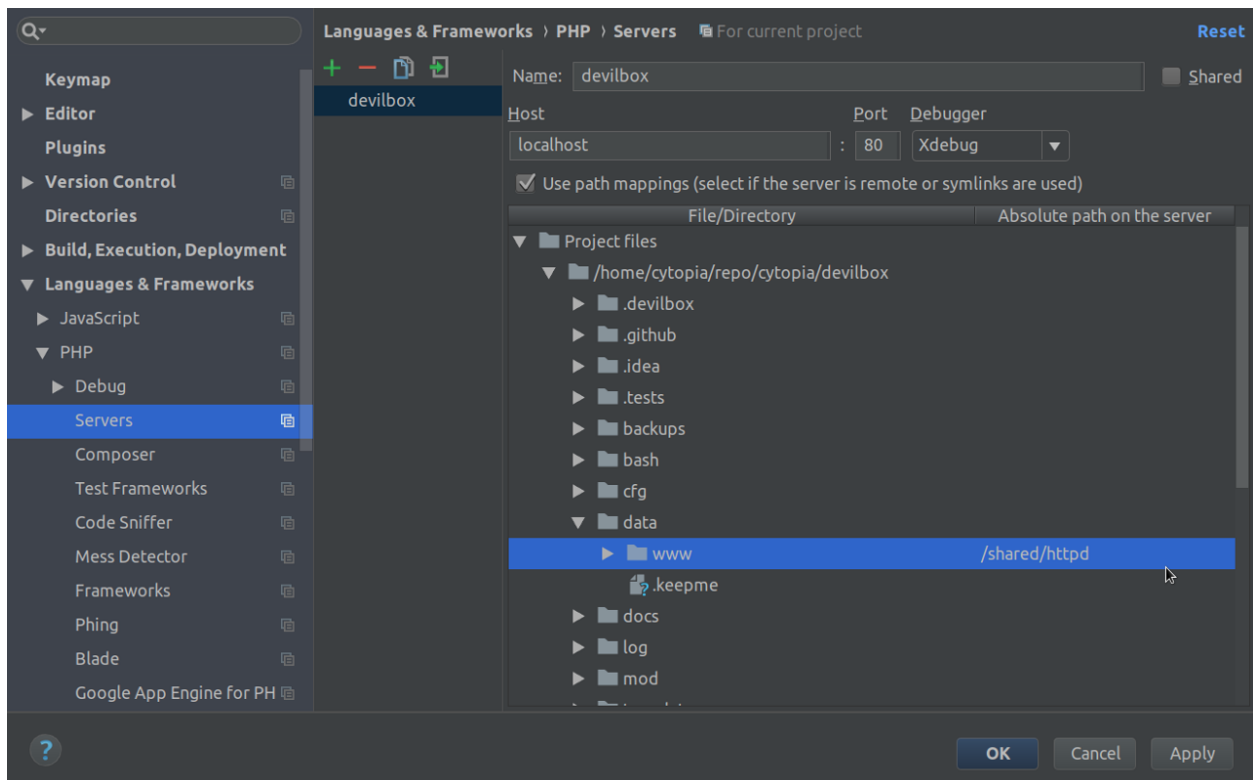
22.2.3 PHPStorm

Linux

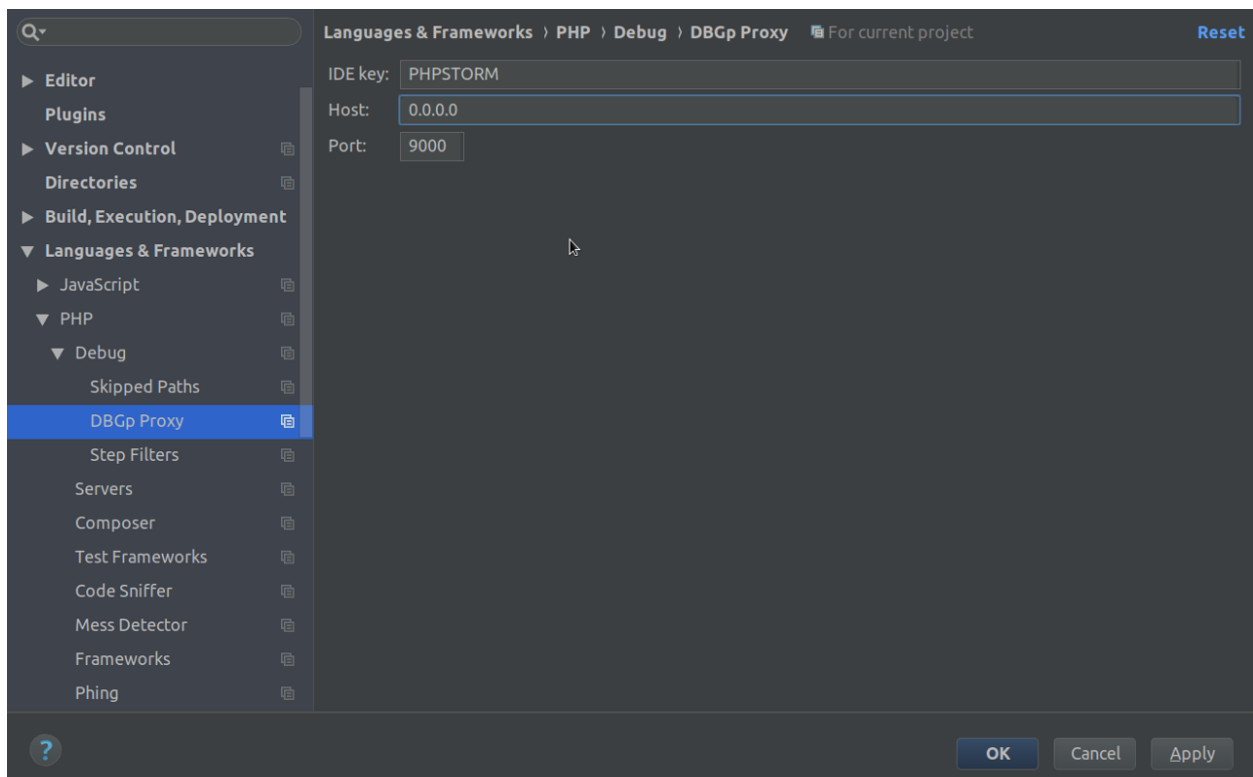
Enable Xdebug for the port set in `xdebug.ini`:



Create a new PHP server and set a path mapping. This tutorial assumes your local Devilbox projects to be in `./data/www` of the Devilbox git directory:



Set DBGp proxy settings:



MacOS (Docker for Mac)

Todo: Help needed. Please provide your config.

MacOS (Docker Toolbox)

Todo: Help needed. Please provide your config.

Windows (Docker for Windows)

Todo: Help needed. Please provide your config.

Windows (Docker Toolbox)

Todo: Help needed. Please provide your config.

22.2.4 Sublime Text 3

1. Install [Xdebug Client](#) via the Sublime Package Control.
2. Configure `Xdebug.sublime-settings` (Tools -> Xdebug -> Settings - User)

Linux

Listing 10: Xdebug-sublime-settings

```
{
  "path_mapping": {
    "/shared/httpd" : "/home/cytopia/repo/devilbox/data/www"
  },
  "url": "",
  "ide_key": "PHPSTORM",
  "host": "0.0.0.0",
  "port": 9000
}
```

MacOS (Docker for Mac)

Todo: Help needed. Please provide your config.

MacOS (Docker Toolbox)

Todo: Help needed. Please provide your config.

Windows (Docker for Windows)

Todo: Help needed. Please provide your config.

Windows (Docker Toolbox)

Todo: Help needed. Please provide your config.

22.2.5 Visual Studio Code

1. Install `vscode-php-debug`
2. Configure `launch.json`

Linux

Listing 11: `launch.json`

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Listen for Xdebug",
      "type": "php",
      "request": "launch",
      "port": 9000,
      "serverSourceRoot": "/shared/httpd",
      "localSourceRoot": "/home/cytopia/repo/devilbox/data/www"
    }, {
      "name": "Launch currently open script",
      "type": "php",
      "request": "launch",
      "program": "${file}",
      "cwd": "${fileDirname}",
      "port": 9000
    }
  ]
}
```

MacOS (Docker for Mac)

Todo: Help needed. Please provide your config.

MacOS (Docker Toolbox)

Todo: Help needed. Please provide your config.

Windows (Docker for Windows)

Todo: Help needed. Please provide your config.

Windows (Docker Toolbox)

Todo: Help needed. Please provide your config.

Custom environment variables

If your application requires a variable to determine if it is run under development or production, you can easily add it and make PHP aware of it.

Table of Contents

- *Add custom environment variables*
- *Use custom environment variables*

23.1 Add custom environment variables

This is fairly simple. Any variable inside the `.env` file is considered an environment variable and automatically known to PHP.

If you for example require a variable `APPLICATION_ENV`, with a value of `production`, you would add the following to the `.env` file:

Listing 1: `.env`

```
APPLICATION_ENV=production
```

You need to restart the Devilbox for the changes to take effect.

Note: There is already a proposed section inside the `.env` file at the very bottom to add you custom variables to differentiate them from the Devilbox required variables.

23.2 Use custom environment variables

Accessing the above defined environment variable on the PHP side is also fairly simple. You can use the PHP's built-in function `getenv` to obtain the value:

Listing 2: index.php

```
<?php
// Example use of getenv()
echo getenv('APPLICATION_ENV');
?>
```

Static Code Analysis

This tutorial gives you a general overview how to do static code analysis from within the PHP container.

See also:

- *Available tools*
- *Work inside the container*

Table of Contents

- *Awesome-ci*
 - *PHPCS*
 - *ESLint*

24.1 Awesome-ci

Awesome-ci is a collection of tools for analysing your workspace and its files. You can for example check for:

- git conflicts
- git ignored files that have not been removed from the git index
- trailing spaces and newlines
- non-utf8 files or utf8 files with bom
- windows line feeds
- null-byte characters
- empty files
- syntax errors for various languages

- inline css or js code
- customized regex

Some of the bundled tools even allow for automatic fixing.

See also:

[awesome-ci](#)

```
# 1. Enter your PHP container
host> ./bash

# 2. Go to your project folder
devilbox@php-7.0.20 $ cd /shared/httpd/my-project

# 3. Run the tools
devilbox@php-7.0.20 $ git-conflicts --path=.
devilbox@php-7.0.20 $ git-ignored --path=.
devilbox@php-7.0.20 $ file-cr --path=.
devilbox@php-7.0.20 $ file-crlf --path=.
devilbox@php-7.0.20 $ file-empty --path=.

# 4. Run tools with more options
devilbox@php-7.0.20 $ syntax-php --path=. --extension=php
devilbox@php-7.0.20 $ syntax-php --path=. --shebang=php

# 5. Various syntax checks
devilbox@php-7.0.20 $ syntax-bash --path=. --text --extension=sh
devilbox@php-7.0.20 $ syntax-css --path=. --text --extension=css
devilbox@php-7.0.20 $ syntax-js --path=. --text --extension=js
devilbox@php-7.0.20 $ syntax-json --path=. --text --extension=json
devilbox@php-7.0.20 $ syntax-markdown --path=. --text --extension=md
devilbox@php-7.0.20 $ syntax-perl --path=. --text --extension=pl
devilbox@php-7.0.20 $ syntax-php --path=. --text --extension=php
devilbox@php-7.0.20 $ syntax-python --path=. --text --extension=python
devilbox@php-7.0.20 $ syntax-ruby --path=. --text --extension=rb
devilbox@php-7.0.20 $ syntax-scss --path=. --text --extension=scss
```

24.2 PHPCS

PHPCS is a code style analyser for PHP.

See also:

[PHPCS](#)

```
# 1. Enter your PHP container
host> ./bash

# 2. Go to your project folder
devilbox@php-7.0.20 $ cd /shared/httpd/my-project

# 3. Run it
devilbox@php-7.0.20 $ phpcs .
```

24.3 ESLint

ESLint is a Javascript static source code analyzer.

See also:

ESLint

```
# 1. Enter your PHP container
host> ./bash

# 2. Go to your project folder
devilbox@php-7.0.20 $ cd /shared/httpd/my-project

# 3. Run it
devilbox@php-7.0.20 $ eslint .
```


This example will use `composer` to install CakePHP from within the PHP container.

See also:

[Official CakePHP Documentation](#)

Table of Contents

- *Overview*
- *Walk through*
 - *1. Enter the PHP container*
 - *2. Create new vhost directory*
 - *3. Install CakePHP*
 - *4. Symlink webroot*
 - *5. Add MySQL Database*
 - *6. Configure database connection*
 - *7. DNS record*
 - *8. Open your browser*

25.1 Overview

The following configuration will be used:

| Project name | VirtualHost directory | Database | TLD_SUFFIX | Project URL |
|--------------|-----------------------|----------|------------|---|
| my-cake | /shared/httpd/my-cake | my_cake | loc | http://my-cake.loc |

25.2 Walk through

It will be ready in eight simple steps:

1. Enter the PHP container
2. Create a new VirtualHost directory
3. Install CakePHP via `composer`
4. Symlink webroot directory
5. Add MySQL database
6. Configure database connection
7. Setup DNS record
8. Visit `http://my-cake.loc` in your browser

See also:

Available tools

25.2.1 1. Enter the PHP container

```
host> ./shell.sh
```

See also:

Work inside the container

25.2.2 2. Create new vhost directory

```
devilbox@php-7.0.20 in /shared/httpd $ mkdir my-cake
```

25.2.3 3. Install CakePHP

```
devilbox@php-7.0.20 in /shared/httpd $ cd my-cake
devilbox@php-7.0.20 in /shared/httpd/my-cake $ composer create-project --prefer-dist _
↪ cakephp/app cakephp
```

25.2.4 4. Symlink webroot

```
devilbox@php-7.0.20 in /shared/httpd/my-cake $ ln -s cakephp/webroot/ htdocs
```

25.2.5 5. Add MySQL Database

```
devilbox@php-7.0.20 in /shared/httpd/my-cake $ mysql -u root -h 127.0.0.1 -p -e
↪ 'CREATE DATABASE my_cake;'
```


25.2.6 6. Configure database connection

```
devilbox@php-7.0.20 in /shared/httpd/my-cake $ vi cakephp/config/app.php
```

Listing 1: cakephp/config/app.php

```
<?php
'Datasources' => [
    'default' => [
        'className' => 'Cake\Database\Connection',
        'driver' => 'Cake\Database\Driver\Mysql',
        'persistent' => false,
        'host' => '127.0.0.1',
        /**
         * CakePHP will use the default DB port based on the driver selected
         * MySQL on MAMP uses port 8889, MAMP users will want to uncomment
         * the following line and set the port accordingly
         */
        //'port' => 'non_standard_port_number',
        'username' => 'root',
        'password' => 'secret',
        'database' => 'my_cake',
        'encoding' => 'utf8',
        'timezone' => 'UTC',
        'flags' => [],
        'cacheMetadata' => true,
    ],
];
```

25.2.7 7. DNS record

If you do not have [Auto-DNS](#) configured, you will need to add the following line to your host operating systems `/etc/hosts` file (or `C:\Windows\System32\drivers\etc` on Windows):

Listing 2: `/etc/hosts`

```
127.0.0.1 my-cake.loc
```

See also:

For in-depth info about adding DNS records on Linux, Windows or MacOS see: [DNS records](#) or [Auto-DNS](#).

25.2.8 8. Open your browser

All set now, you can visit <http://my-cake.loc> in your browser.

This example will use `drush` to install Drupal from within the PHP container.

See also:

[Official Drupal Documentation](#)

Table of Contents

- *Overview*
- *Walk through*
 - *1. Enter the PHP container*
 - *2. Create new vhost directory*
 - *3. Install Drupal*
 - *4. Symlink webroot*
 - *5. DNS record*
 - *6. Open your browser*

26.1 Overview

The following configuration will be used:

| Project name | VirtualHost directory | Database | TLD_SUFFIX | Project URL |
|--------------|-------------------------|-----------|------------|---|
| my-drupal | /shared/httpd/my-drupal | my_drupal | loc | http://my-drupal.loc |

26.2 Walk through

It will be ready in six simple steps:

1. Enter the PHP container
2. Create a new VirtualHost directory
3. Install Drupal via drush
4. Symlink webroot directory
5. Setup DNS record
6. Visit <http://my-drupal.loc> in your browser

See also:

Available tools

26.2.1 1. Enter the PHP container

```
host> ./shell.sh
```

See also:

Work inside the container

26.2.2 2. Create new vhost directory

```
devilbox@php-7.0.20 in /shared/httpd $ mkdir my-drupal
```

26.2.3 3. Install Drupal

```
devilbox@php-7.0.20 in /shared/httpd $ cd my-drupal
devilbox@php-7.0.20 in /shared/httpd/my-drupal $ drush dl drupal
```

26.2.4 4. Symlink webroot

```
devilbox@php-7.0.20 in /shared/httpd/my-drupal $ ln -s drupal-8.3.3/ htdocs
```

26.2.5 5. DNS record

If you do not have *Auto-DNS* configured, you will need to add the following line to your host operating systems `/etc/hosts` file (or `C:\Windows\System32\drivers\etc` on Windows):

Listing 1: `/etc/hosts`

```
127.0.0.1 my-drupal.loc
```

See also:

For in-depth info about adding DNS records on Linux, Windows or MacOS see: *DNS records* or *Auto-DNS*.

26.2.6 6. Open your browser

Open your browser at <http://my-drupal.loc> and follow the Drupal installation steps.

Note: When asked about MySQL hostname, choose `127.0.0.1`.

This example will install Joomla from within the PHP container.

See also:

[Official Joomla Documentation](#)

Table of Contents

- *Overview*
- *Walk through*
 - *1. Enter the PHP container*
 - *2. Create new vhost directory*
 - *3. Download and extract Joomla*
 - *4. Symlink webroot*
 - *5. DNS record*
 - *6. Open your browser*

27.1 Overview

The following configuration will be used:

| Project name | VirtualHost directory | Database | TLD_SUFFIX | Project URL |
|--------------|-------------------------|----------|------------|---|
| my-joomla | /shared/httpd/my-joomla | n.a. | loc | http://my-joomla.loc |

27.2 Walk through

It will be ready in six simple steps:

1. Enter the PHP container
2. Create a new VirtualHost directory
3. Download and extract Joomla
4. Symlink webroot directory
5. Setup DNS record
6. Visit <http://my-joomla.loc> in your browser

See also:

Available tools

27.2.1 1. Enter the PHP container

```
host> ./shell.sh
```

See also:

Work inside the container

27.2.2 2. Create new vhost directory

```
devilbox@php-7.0.20 in /shared/httpd $ mkdir my-joomla
```

27.2.3 3. Download and extract Joomla

```
devilbox@php-7.0.20 in /shared/httpd $ cd my-joomla
devilbox@php-7.0.20 in /shared/httpd/my-joomla $ wget -O joomla.tar.gz https://
↪downloads.joomla.org/cms/joomla3/3-8-0/joomla_3-8-0-stable-full_package-tar-gz?
↪format=gz
devilbox@php-7.0.20 in /shared/httpd $ mkdir joomla
devilbox@php-7.0.20 in /shared/httpd $ tar xvfz joomla.tar.gz -C joomla/
```

27.2.4 4. Symlink webroot

```
devilbox@php-7.0.20 in /shared/httpd/my-joomla $ ln -s joomla/ htdocs
```

27.2.5 5. DNS record

If you do not have *Auto-DNS* configured, you will need to add the following line to your host operating systems /etc/hosts file (or C:\Windows\System32\drivers\etc on Windows):

Listing 1: /etc/hosts

```
127.0.0.1 my-joomla.loc
```

See also:

For in-depth info about adding DNS records on Linux, Windows or MacOS see: [DNS records](#) or [Auto-DNS](#).

27.2.6 6. Open your browser

Open your browser at <http://my-joomla.loc>

CHAPTER 28

Setup Laravel

This example will use `laravel` to install Laravel from within the PHP container.

See also:

[Official Laravel Documentation](#)

Table of Contents

- *Overview*
- *Walk through*
 - *1. Enter the PHP container*
 - *2. Create new vhost directory*
 - *3. Install Laravel*
 - *4. Symlink webroot*
 - *5. DNS record*
 - *6. Open your browser*

28.1 Overview

The following configuration will be used:

| Project name | VirtualHost directory | Database | TLD_SUFFIX | Project URL |
|--------------|--------------------------|----------|------------|---|
| my-laravel | /shared/httpd/my-laravel | n.a. | loc | http://my-laravel.loc |

28.2 Walk through

It will be ready in six simple steps:

1. Enter the PHP container
2. Create a new VirtualHost directory
3. Install Laravel
4. Symlink webroot directory
5. Setup DNS record
6. Visit <http://my-laravel.loc> in your browser

See also:

Available tools

28.2.1 1. Enter the PHP container

```
host> ./shell.sh
```

See also:

Work inside the container

28.2.2 2. Create new vhost directory

```
devilbox@php-7.0.20 in /shared/httpd $ mkdir my-laravel
```

28.2.3 3. Install Laravel

```
devilbox@php-7.0.20 in /shared/httpd $ cd my-laravel
devilbox@php-7.0.20 in /shared/httpd/my-laravel $ laravel new laravel-project
```

28.2.4 4. Symlink webroot

```
devilbox@php-7.0.20 in /shared/httpd/my-laravel $ ln -s laravel-project/public/ htdocs
```

28.2.5 5. DNS record

If you do not have [Auto-DNS](#) configured, you will need to add the following line to your host operating systems `/etc/hosts` file (or `C:\Windows\System32\drivers\etc` on Windows):

Listing 1: `/etc/hosts`

```
127.0.0.1 my-laravel.loc
```

See also:

For in-depth info about adding DNS records on Linux, Windows or MacOS see: [DNS records](#) or [Auto-DNS](#).

28.2.6 6. Open your browser

Open your browser at <http://my-laravel.loc>

This example will use `phalcon` to install Phalcon from within the PHP container.

See also:

[Official Phalcon Documentation](#)

Table of Contents

- *Overview*
- *Walk through*
 - *1. Enter the PHP container*
 - *2. Create new vhost directory*
 - *3. Install Phalcon*
 - *4. Symlink webroot*
 - *5. DNS record*
 - *6. Open your browser*

29.1 Overview

The following configuration will be used:

| Project name | VirtualHost directory | Database | TLD_SUFFIX | Project URL |
|--------------|--------------------------|----------|------------|---|
| my-phalcon | /shared/httpd/my-phalcon | n.a. | loc | http://my-phalcon.loc |

29.2 Walk through

It will be ready in six simple steps:

1. Enter the PHP container
2. Create a new VirtualHost directory
3. Install Phalcon
4. Symlink webroot directory
5. Setup DNS record
6. Visit <http://my-phalcon.loc> in your browser

See also:

Available tools

29.2.1 1. Enter the PHP container

```
host> ./shell.sh
```

See also:

Work inside the container

29.2.2 2. Create new vhost directory

```
devilbox@php-7.0.20 in /shared/httpd $ mkdir my-phalcon
```

29.2.3 3. Install Phalcon

```
devilbox@php-7.0.20 in /shared/httpd $ cd my-phalcon
devilbox@php-7.0.20 in /shared/httpd/my-phalcon $ phalcon project phalconphp
```

29.2.4 4. Symlink webroot

```
devilbox@php-7.0.20 in /shared/httpd/my-phalcon $ ln -s phalconphp/public/ htdocs
```

29.2.5 5. DNS record

If you do not have [Auto-DNS](#) configured, you will need to add the following line to your host operating systems `/etc/hosts` file (or `C:\Windows\System32\drivers\etc` on Windows):

Listing 1: `/etc/hosts`

```
127.0.0.1 my-phalcon.loc
```

See also:

For in-depth info about adding DNS records on Linux, Windows or MacOS see: [DNS records](#) or [Auto-DNS](#).

29.2.6 6. Open your browser

Open your browser at <http://my-phalcon.loc>

CHAPTER 30

Setup Symfony

This example will use `symfony` to install Symfony from within the PHP container.

See also:

[Official Symfony Documentation](#)

Table of Contents

- *Overview*
- *Walk through*
 - *1. Enter the PHP container*
 - *2. Create new vhost directory*
 - *3. Install Symfony*
 - *4. Symlink webroot*
 - *5. Enable Symfony prod (`app.php`)*
 - *6. DNS record*
 - *7. Open your browser*

30.1 Overview

The following configuration will be used:

| Project name | VirtualHost directory | Database | TLD_SUFFIX | Project URL |
|--------------|--------------------------|----------|------------|---|
| my-symfony | /shared/httpd/my-symfony | n.a. | loc | http://my-symfony.loc |

30.2 Walk through

It will be ready in seven simple steps:

1. Enter the PHP container
2. Create a new VirtualHost directory
3. Install Symfony
4. Symlink webroot directory
5. Enable Symfony prod (`app.php`)
6. Setup DNS record
7. Visit <http://my-symfony.loc> in your browser

See also:

Available tools

30.2.1 1. Enter the PHP container

```
host> ./shell.sh
```

See also:

Work inside the container

30.2.2 2. Create new vhost directory

```
devilbox@php-7.0.20 in /shared/httpd $ mkdir my-symfony
```

30.2.3 3. Install Symfony

```
devilbox@php-7.0.20 in /shared/httpd $ cd my-symfony
devilbox@php-7.0.20 in /shared/httpd/my-symfony $ symfony new symfony
```

30.2.4 4. Symlink webroot

```
devilbox@php-7.0.20 in /shared/httpd/my-symfony $ ln -s symfony/web/ htdocs
```

30.2.5 5. Enable Symfony prod (`app.php`)

```
devilbox@php-7.0.20 in /shared/httpd/my-symfony $ cd symfony/web
devilbox@php-7.0.20 in /shared/httpd/my-symfony/symfony/web $ ln -s app.php index.php
```

30.2.6 6. DNS record

If you do not have *Auto-DNS* configured, you will need to add the following line to your host operating systems `/etc/hosts` file (or `C:\Windows\System32\drivers\etc` on Windows):

Listing 1: `/etc/hosts`

```
127.0.0.1 my-symfony.loc
```

See also:

For in-depth info about adding DNS records on Linux, Windows or MacOS see: *DNS records* or *Auto-DNS*.

30.2.7 7. Open your browser

Open your browser at <http://my-symfony.loc>

Setup Wordpress

This example will use `git` to install Wordpress from within the PHP container.

See also:

[Official Wordpress Documentation](#)

Table of Contents

- *Overview*
- *Walk through*
 - *1. Enter the PHP container*
 - *2. Create new vhost directory*
 - *3. Download Wordpress via `git`*
 - *4. Symlink webroot*
 - *5. DNS record*
 - *6. Open your browser*

31.1 Overview

The following configuration will be used:

| Project name | VirtualHost directory | Database | TLD_SUFFIX | Project URL |
|--------------|-----------------------|----------|------------|---|
| my-wp | /shared/httpd/my-wp | my_wp | loc | http://my-wp.loc |

31.2 Walk through

It will be ready in six simple steps:

1. Enter the PHP container
2. Create a new VirtualHost directory
3. Download Wordpress via `git`
4. Symlink webroot directory
5. Setup DNS record
6. Visit `http://my-wp.loc` in your browser

See also:

Available tools

31.2.1 1. Enter the PHP container

```
host> ./shell.sh
```

See also:

Work inside the container

31.2.2 2. Create new vhost directory

```
devilbox@php-7.0.20 in /shared/httpd $ mkdir my-wp
```

31.2.3 3. Download Wordpress via `git`

```
devilbox@php-7.0.20 in /shared/httpd $ cd my-wp
devilbox@php-7.0.20 in /shared/httpd/my-wp $ git clone https://github.com/WordPress/
↳WordPress wordpress.git
```

31.2.4 4. Symlink webroot

```
devilbox@php-7.0.20 in /shared/httpd/my-wp $ ln -s wordpress.git/ htdocs
```

31.2.5 5. DNS record

If you do not have *Auto-DNS* configured, you will need to add the following line to your host operating systems `/etc/hosts` file (or `C:\Windows\System32\drivers\etc` on Windows):

Listing 1: /etc/hosts

```
127.0.0.1 my-wp.loc
```

See also:

For in-depth info about adding DNS records on Linux, Windows or MacOS see: [DNS records](#) or [Auto-DNS](#).

31.2.6 6. Open your browser

Open your browser at <http://my-wp.loc>

This example will use `composer` to install Yii from within the PHP container.

See also:

[Official Yii Documentation](#)

Table of Contents

- *Overview*
- *Walk through*
 - *1. Enter the PHP container*
 - *2. Create new vhost directory*
 - *3. Install Yii2 via `composer`*
 - *4. Symlink webroot*
 - *5. DNS record*
 - *6. Open your browser*

32.1 Overview

The following configuration will be used:

| Project name | VirtualHost directory | Database | TLD_SUFFIX | Project URL |
|--------------|-----------------------|----------|------------|---|
| my-yii | /shared/httpd/my-yii | n.a. | loc | http://my-yii.loc |

32.2 Walk through

It will be ready in six simple steps:

1. Enter the PHP container
2. Create a new VirtualHost directory
3. Install Yii2 via `composer`
4. Symlink webroot directory
5. Setup DNS record
6. Visit `http://my-wp.loc` in your browser

See also:

Available tools

32.2.1 1. Enter the PHP container

```
host> ./shell.sh
```

See also:

Work inside the container

32.2.2 2. Create new vhost directory

```
devilbox@php-7.0.20 in /shared/httpd $ mkdir my-yii
```

32.2.3 3. Install Yii2 via `composer`

```
devilbox@php-7.0.20 in /shared/httpd $ cd my-yii
devilbox@php-7.0.20 in /shared/httpd/my-yii $ composer create-project --prefer-dist --
↪stability=dev yiisoft/yii2-app-basic yii2-dev
```

32.2.4 4. Symlink webroot

```
devilbox@php-7.0.20 in /shared/httpd/my-yii $ ln -s yii2-dev/web/ htdocs
```

32.2.5 5. DNS record

If you do not have *Auto-DNS* configured, you will need to add the following line to your host operating systems `/etc/hosts` file (or `C:\Windows\System32\drivers\etc` on Windows):

Listing 1: /etc/hosts

```
127.0.0.1 my-yii.loc
```

See also:

For in-depth info about adding DNS records on Linux, Windows or MacOS see: [DNS records](#) or [Auto-DNS](#).

32.2.6 6. Open your browser

Open your browser at <http://my-yii.loc>

This example will use `composer` to install Zend from within the PHP container.

See also:

[Official Zend Documentation](#)

Table of Contents

- *Overview*
- *Walk through*
 - *1. Enter the PHP container*
 - *2. Create new vhost directory*
 - *3. Install Zend via `composer`*
 - *4. Symlink webroot*
 - *5. DNS record*
 - *6. Open your browser*

33.1 Overview

The following configuration will be used:

| Project name | VirtualHost directory | Database | TLD_SUFFIX | Project URL |
|--------------|-----------------------|----------|------------|---|
| my-zend | /shared/httpd/my-zend | n.a. | loc | http://my-zend.loc |

33.2 Walk through

It will be ready in six simple steps:

1. Enter the PHP container
2. Create a new VirtualHost directory
3. Install Zend via `composer`
4. Symlink webroot directory
5. Setup DNS record
6. Visit `http://my-wp.loc` in your browser

See also:

Available tools

33.2.1 1. Enter the PHP container

```
host> ./shell.sh
```

See also:

Work inside the container

33.2.2 2. Create new vhost directory

```
devilbox@php-7.0.20 in /shared/httpd $ mkdir my-zend
```

33.2.3 3. Install Zend via `composer`

```
devilbox@php-7.0.20 in /shared/httpd $ cd my-zend
devilbox@php-7.0.20 in /shared/httpd/my-zend $ composer create-project --prefer-dist_
↪ zendframework/skeleton-application zend
```

33.2.4 4. Symlink webroot

```
devilbox@php-7.0.20 in /shared/httpd/my-zend $ ln -s zend/public/ htdocs
```

33.2.5 5. DNS record

If you do not have *Auto-DNS* configured, you will need to add the following line to your host operating systems `/etc/hosts` file (or `C:\Windows\System32\drivers\etc` on Windows):

Listing 1: /etc/hosts

```
127.0.0.1 my-zend.loc
```

See also:

For in-depth info about adding DNS records on Linux, Windows or MacOS see: [DNS records](#) or [Auto-DNS](#).

33.2.6 6. Open your browser

Open your browser at <http://my-zend.loc>

DNS records

Project DNS records are required, because each project is using its own virtual host with its own unique server name. The server name is constructed by a `<project-directory>` and the *TLD_SUFFIX* and requires the same DNS record to be present in order to access it.

See also:

This section gives you an overview about how to create separate DNS records for each project. It has to be done for each project, however if you want to automate the process, refer to *Auto-DNS*.

Table of Contents


- *Examples*
- *Creating DNS records*
 - *Native Docker*
 - * *Linux*
 - * *MacOS*
 - * *Windows*
 - *Docker Toolbox*
 - * *MacOS*
 - * *Windows*
- *Verify*

34.1 Examples

In order to better illustrate the process, we are going to use two projects as an example. See the following table for project directories and TLD_SUFFIX.

| Project directory | TLD_SUFFIX | Project URL | Required DNS name |
|-------------------|------------|---|-------------------|
| project-1 | loc | http://project-1.loc | project-1.loc |
| www.project-1 | loc | http://www.project-1.loc | www.project-1.loc |

Note: When you have created the above two projects, you can check the vhost page on the Devilbox intranet. It will tell you exactly what DNS record to add.

 devilbox Home Virtual Hosts Emails Databases ▾ Info ▾ Tools ▾

Virtual Hosts

| Project | DocumentRoot | Valid | URL |
|-----------|----------------------------|-------|--|
| project-1 | /data/www/project-1/htdocs | ERR | No Host DNS record found. Add the following to <code>/etc/hosts</code> : <code>127.0.0.1 project-1.loc</code> |

Render time: 0.01 sec Github Credits Debug (0)

Important: The IP address `127.0.0.1` is different for *Docker Toolbox*

34.2 Creating DNS records

When creating DNS records for your host operating system, there are two distinctions to be made. If you use Native Docker (the default and recommended Docker), you can always use `127.0.0.1` as your IP address for the DNS record. If however you use Docker Toolbox, you first need to find out the IP address of the Docker Toolbox virtual machine.

See also:

Docker Toolbox

34.2.1 Native Docker

Linux

Use your favorite editor and open `/etc/hosts` with root privileges. The following example uses vim to add the two example DNS records.

```
host> sudo vim /etc/hosts

127.0.0.1 project-1.loc
127.0.0.1 www.project-1.loc
```

MacOS

Use your favorite editor and open `/etc/hosts` with root privileges. The following example uses vim to add the two example DNS records.

```
host> sudo vim /etc/hosts

127.0.0.1 project-1.loc
127.0.0.1 www.project-1.loc
```

Windows

On Windows you need to open `C:\Windows\System32\drivers\etc` with administrative privileges and add the following two lines:

```
127.0.0.1 project-1.loc
127.0.0.1 www.project-1.loc
```

34.2.2 Docker Toolbox

When using Docker Toolbox the Devilbox runs inside a virtual machine and therefore the webserver port (80) is not exposed to your host operating system. So your DNS record must point to the virtual machine instead of your host system.

1. Find out the IP address the virtual machine is running on
2. Add a DNS entry to your host operating system for this IP address.

For the sake of this example, let's assume the virtual machine is running on `192.16.0.1`

MacOS

Use your favorite editor and open `/etc/hosts` with root privileges. The following example uses vim to add the two example DNS records.

```
host> sudo vim /etc/hosts

192.16.0.1 project-1.loc
192.16.0.1 www.project-1.loc
```

Windows

On Windows you need to open `C:\Windows\System32\drivers\etc` with administrative privileges and add the following two lines:

```
192.16.0.1 project-1.loc
192.16.0.1 www.project-1.loc
```

34.3 Verify

After settings the DNS records, you can use the `ping` command to verify if everything works.

```
host> ping -c1 project-1.loc

PING project-1.loc (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.066 ms
```

```
host> ping -c1 www.project-1.loc

PING www.project-1.loc (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.066 ms
```

Customized virtual host (vhost-gen)

Table of Contents

- *vhost-gen*
 - *What is vhost-gen*
 - *Where do I find templates*
 - *How does it work*
 - *How to apply templates to a specific project*
 - * *1. Retrieve or set template directory value*
 - * *2. Copy webserver template to project template directory*
 - * *3. Adjust template*
 - * *4. Make Devilbox pick up those changes*
- *Templates explained*
 - *Ensure yaml files are valid*
 - *Template variables*
 - * *Global variables*
 - * *vHost type variable*
 - * *Feature variables*
 - *Template structure*
 - * *1. vhost:*
 - * *2. vhost_type:*
 - * *3. features:*

- *Apply Changes*
 - *Rename project directory*
 - *Restart the Devilbox*
- *Further readings*

35.1 vhost-gen

35.1.1 What is vhost-gen

`vhost-gen` is a python script which is able to dynamically generate Apache 2.2, Apache 2.4 and Nginx virtual host or reverse proxy configuration files.

It is intended to be used by other means of automation such as change of directories or change of listening ports.

See also:

If you intend to use `vhost-gen` for your own projects, have a look at its project page and its sister projects:

- `vhost-gen`
- `watcherd`
- `watcherp`

35.1.2 Where do I find templates

The latest version of `vhost-gen` templates are shipped in the Devilbox git directory under `templates/vhost-gen`. You can however also download them directly from its own git directory.

See also:

<https://github.com/devilbox/vhost-gen/tree/master/etc/templates>.

35.1.3 How does it work

By default new virtual hosts are automatically generated and enabled by `vhost-gen` and `watcherp` using the vanilla templates which are glued into the webserver Docker images. The used templates are exactly the same as what you will find in `templates/vhost-gen`.

This ensures to have equal and sane default virtual host for all of your projects. If you want to have a different virtual host configuration for a specific project of yours, you can copy a corresponding template into your project directory and adjust it to your needs.

35.1.4 How to apply templates to a specific project

Customizing a virtual host via `vhost-gen` template is generally done in four steps:

1. Retrieve or set template directory value in `.env`.
2. Copy webserver template to project template directory
3. Adjust template

4. Make Devilbox pick up those changes

Let's assume the following default values and one project named `project-1`:

| Variable | Value |
|---------------------------------------|---|
| Devilbox path | /home/user/devilbox |
| Templates to copy from | /home/user/devilbox/templates/vhost-gen |
| Project name | project-1 |
| <code>HTTPD_TEMPLATE_DIR</code> | .devilbox (default value) |
| <code>HOST_PATH_HTTPD_DATA_DIR</code> | ./data/www (default value) |

Those assumed settings will result in the following directory paths which must be created by you:

| What | Path |
|------------------------|---|
| Project directory path | /home/user/devilbox/data/www/project-1/ |
| Project template path | /home/user/devilbox/data/www/project-1/.devilbox/ |

1. Retrieve or set template directory value

By default the `HTTPD_TEMPLATE_DIR` value is `.devilbox`. This is defined in the `.env` file. Feel free to change it to whatever directory name you prefer, but keep in mind that it will change the *Project template path* which you need to create yourself.

For this example we will keep the default value for the sake of simplicity: `.devilxbox`.

Note: The `HTTPD_TEMPLATE_DIR` value is a global setting and will affect all projects.

2. Copy webserver template to project template directory

First you need to ensure that the `HTTPD_TEMPLATE_DIR` exists within you project.

```
# Navigate to the Devilbox directory
host> cd /home/user/devilbox

# Create template directory in your project
host> mkdir ./data/www/project-1/.devilbox
```

Then you can copy the templates.

```
host> cp templates/vhost-gen/* ./data/www/project-1/.devilbox
```

Note: You actually only need to copy the template of your chosen webserver (either Apache 2.2, Apache 2.4 or Nginx), however it is good practice to copy all templates and also adjust all templates synchronously. This allows you to change web server versions and still keep your virtual host settings.

3. Adjust template

At this stage you can start adjusting the template. Either do that for the webserver version you have enabled via `HTTPD_SERVER`: `/home/user/devilbox/data/www/project-1/.devilbox/apache22.`

yaml. `/home/user/devilbox/data/www/project-1/.devilbox/apache24.yaml`, `/home/user/devilbox/data/www/project-1/.devilbox/nginx.yaml` or do it for all of them synchronously.

Note: What exactly to change will be explained later.

4. Make Devilbox pick up those changes

Whenever you change a project vhost template or the `HTTPD_TEMPLATE_DIR` value, you need to restart the Devilbox.

Note: It is also possible to do it without a restart which will be explained later.

35.2 Templates explained

Before the templates are explained, have a look at the following table to find out what template needs to be in place for what webserver version.

| Webserver | Template |
|----------------|----------------------------|
| Apache 2.2 | <code>apache22.yaml</code> |
| Apache 2.4 | <code>apache22.yaml</code> |
| Nginx stable | <code>nginx.yaml</code> |
| Nginx mainline | <code>nginx.yaml</code> |

Note: Nginx stable and mainline share the same template as their syntax has no special differences, whereas Apache 2.2 and Apache 2.4 have slight differences in syntax and therefore require two different templates.

35.2.1 Ensure yaml files are valid

Warning: Pay close attention that you do not use TAB (`\t`) characters for indenting the vhost-gen yaml files. Some editors might automatically indent using TABs, so ensure they are replaced with spaces. If TAB characters are present, those files become invalid and won't work. <https://github.com/cytopia/devilbox/issues/142>

You can use the bundled `yamllint` binary inside the container to validate your config.

```
# Navigate to the Devilbox directory
host> cd /home/user/devilbox

# Enter the PHP container
host> ./shell.sh

# Go to your project's template directory
devilbox@php-7.0.19 in /shared/httpd $ cd project-1/.devilbox
```

(continues on next page)

(continued from previous page)

```
# Check the syntax of apache22.yml
devilbox@php-7.0.19 in /shared/httpd/project-1/.devilbox $ yamllint apache22.yml

108:81 error line too long (90 > 80 characters) (line-length)
139:81 error line too long (100 > 80 characters) (line-length)
140:81 error line too long (84 > 80 characters) (line-length)
142:81 error line too long (137 > 80 characters) (line-length)
```

Long line errors can safely be ignored.

35.2.2 Template variables

Every uppercase string which begins with `__` and ends by `__` (such as `__PORT__`) is a variable that will be replaced by a value. Variables can contain a string, a multi-line string or can also be replaced to an empty value.

Global variables

There are *global variables* that are determined by the command line arguments of `vhost-gen` itself or are elsewhere replaced by the Devilbox webserver container such as:

- `__PORT__`
- `__DEFAULT_VHOST__`
- `__VHOST_NAME__`
- `__ACCESS_LOG__`
- `__ERROR_LOG__`

vHost type variable

There are also two variables that will be replaced according to the type of the vhost - either a normal vhost or a reverse proxy vhost.

- `__VHOST_DOCROOT__`
- `__VHOST_PROXY__`

The Devilbox always uses a normal vhost by default, so the `__VHOST_DOCROOT__` variable will be replaced by what the `vhost_type.docroot` section provides. The `vhost_type.rproxy` will be ignored and `__VHOST_PROXY__` will be replaced by an empty string.

Feature variables

All other variables will be replaced by what is provided in the `features:` section. All subsections of `features:` have corresponding variables in the following form:

| Feature directive | Variable name pattern |
|--------------------------|-----------------------------|
| <code>lower_case:</code> | <code>__UPPER_CASE__</code> |

As an example, the contents of the `features.php_fpm:` section will be replaced into the `__PHP_FPM__` variable.

35.2.3 Template structure

Each vhost-gen template has three main yaml directives:

1. `vhost:`
2. `vhost_type:`
3. `features:`

1. `vhost:`

The `vhost:` directive will contain the final resulting virtual host configuration that will be applied by the webserver. Each of its containing variables will be substituted and its content will be copied to a webserver configuration file.

By default the `vhost:` section has variables from global scope, from the `vhost_type:` section and from the `features:` section.

You can also fully hard-code your webserver configuration without any variables. This way you can specify a fully self-brewed webserver configuration. An example for Apache 2.2 could look like this:

```
vhost: |
  <VirtualHost *:80>
    ServerName    example.com

    CustomLog      "/var/log/apache/access.log" combined
    ErrorLog       "/var/log/apache/error.log"

    DocumentRoot  "/shared/httpd/project-1/htdocs"
    <Directory "/shared/httpd/project-1/htdocs">
      DirectoryIndex index.php

      AllowOverride All
      Options All

      RewriteEngine on
      RewriteBase /

      Order allow,deny
      Allow from all
    </Directory>

    ProxyPassMatch ^/(.*\.php(/.*)?)$ fcgi://127.0.0.1:9000/shared/httpd/project-1/
    ↪htdocs/$1
  </VirtualHost>
```

2. `vhost_type:`

The `vhost_type:` contains `docroot` and `rproxy`. The Devilbox only makes use of `docroot` which holds the definition of a normal vhost. Its content will be replaced into the `__VHOST_DOCROOT__` variable.

The `rproxy` section will be ignored and the `__VHOST_RPROXY__` variable will contain an empty value.

| vHost Type section | Variable to be replaced into |
|-----------------------|---------------------------------------|
| <code>docroot:</code> | <code>__VHOST_DOCROOT__</code> |
| <code>rproxy:</code> | <code>__VHOST_RPROXY__</code> (empty) |

3. features :

This section contains directives that will all be replaced into `vhost :` variables.

| Feature section | Variable to be replaced into |
|-------------------------------|--------------------------------|
| <code>php_fpm:</code> | <code>__PHP_FPM__</code> |
| <code>alias:</code> | <code>__ALIASES__</code> |
| <code>deny:</code> | <code>__DENIES__</code> |
| <code>server_status:</code> | <code>__SERVER_STATUS__</code> |
| <code>xdomain_request:</code> | <code>__XDOMAIN_REQ__</code> |

35.3 Apply Changes

After having edited your `vhost-gen` template files, you still need to apply these changes. This can be achieved in two ways:

1. Rename your project directory back and forth
2. Restart the Devilbox

35.3.1 Rename project directory

```
# Navigate to the data directory
host> /home/user/devilbox/data/www

# Rename your project to something else
host> mv project-1 project-1.tmp

# Rename your project to its original name
host> mv project-1.tmp project-1
```

If you want to understand what is going on right now, check the docker logs for the web server.

```
# Navigate to the devilbox directory
host> /home/user/devilbox

# Check docker logs
host> docker-compose logs httpd

httpd_1 | vhostgen: [2018-03-18 11:46:52] Adding: project-1.tmp.loc
httpd_1 | watcherd: [2018-03-18 11:46:52] [OK] ADD: succeeded: /shared/httpd/
↪project-1.tmp
httpd_1 | watcherd: [2018-03-18 11:46:52] [OK] DEL: succeeded: /shared/httpd/
↪project-1
httpd_1 | watcherd: [2018-03-18 11:46:52] [OK] TRIGGER succeeded: /usr/local/
↪apache2/bin/httpd -k restart

httpd_1 | vhostgen: [2018-03-18 11:46:52] Adding: project-1loc
httpd_1 | watcherd: [2018-03-18 11:46:52] [OK] ADD: succeeded: /shared/httpd/
↪project-1
httpd_1 | watcherd: [2018-03-18 11:46:52] [OK] DEL: succeeded: /shared/httpd/
↪project-1.tmp
httpd_1 | watcherd: [2018-03-18 11:46:52] [OK] TRIGGER succeeded: /usr/local/
↪apache2/bin/httpd -k restart
```

What happened?

The directory changes have been noticed and a new virtual host has been created. This time however your new vhost-gen template has been read and the changes have applied.

Note: Renaming a project directory will only affect a single project. In case you change the value of `HTTPD_TEMPLATE_DIR` it will affect all projects and you would have to rename all project directories. In this case it is much faster to just restart the Devilbox.

35.3.2 Restart the Devilbox

Stop the Devilbox and start it up again.

35.4 Further readings

See also:

Have a look at the following examples which involve customizing vhost-gen templates:

- *Adding Sub domains*

CHAPTER 36

HTTPS (SSL)

This page shows you how to use the Devilbox on https and how to import the Certificate Authority into your browser once, so that you always and automatically get valid SSL certificates for all new projects.

SSL certificates are generated automatically and there is nothing to do from your side.

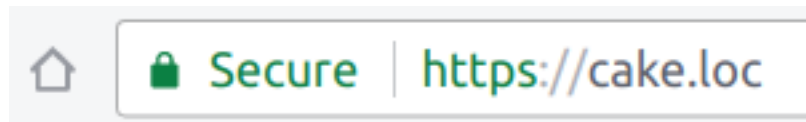


Table of Contents

- *TL;DR*
- *How does it work*
 - *Certificate Authority*
 - *SSL Certificates*
- *Import the CA into your browser*
 - *Chrome / Chromium*
 - *Firefox*
- *Further Reading*

36.1 TL;DR

Import the Certificate Authority into your browser and you are all set.

36.2 How does it work

36.2.1 Certificate Authority

When the Devilbox starts up for the first time, it will generate a [Certificate Authority](#) and will store its public and private key in `./ca/` within the Devilbox git directory.

The keys are only generated if they don't exist and kept permanently if you don't delete them manually, i.e. they are not overwritten.

```
host> cd path/to/devilbox
host> ls -l ca/
-rw-r--r--  1 cytopia cytopia 1558 May  2 11:12 devilbox-ca.crt
-rw-----  1 cytopia cytopia 1675 May  2 11:12 devilbox-ca.key
-rw-r--r--  1 cytopia cytopia  17 May  4 08:35 devilbox-ca.srl
```

36.2.2 SSL Certificates

Whenever you create a new project directory, multiple things happen in the background:

1. A new virtual host is created
2. DNS is provided via [Auto-DNS](#)
3. A new SSL certificate is generated for that vhost
4. **The SSL certificate is signed by the Devilbox Certificate Authority**

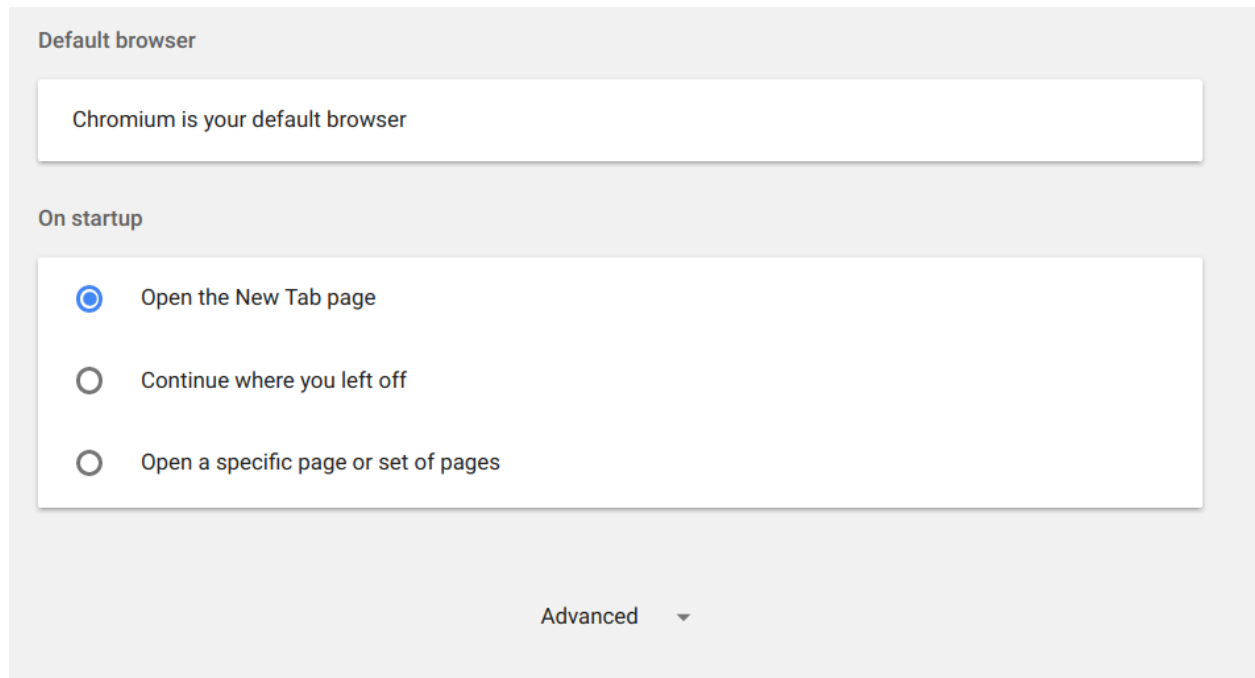
By having a SSL certificates signed by the provided CA, you will only have to import the CA into your browser ones and all current projects and future projects will automatically have valid and trusted SSL certificates without any further work.

Important: Importing the CA into the browser is also recommended and required for the Devilbox intranet page to work properly.

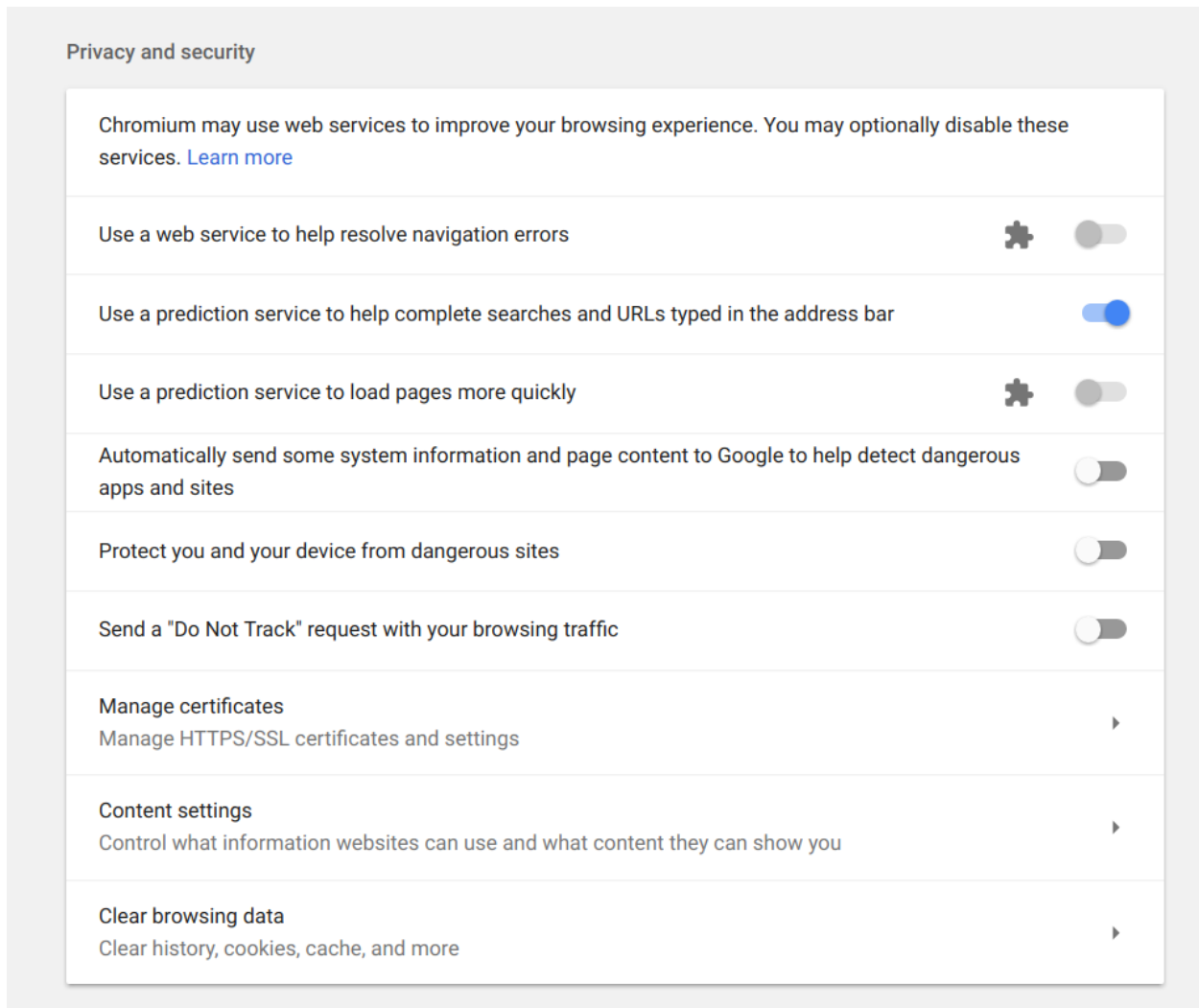
36.3 Import the CA into your browser

36.3.1 Chrome / Chromium

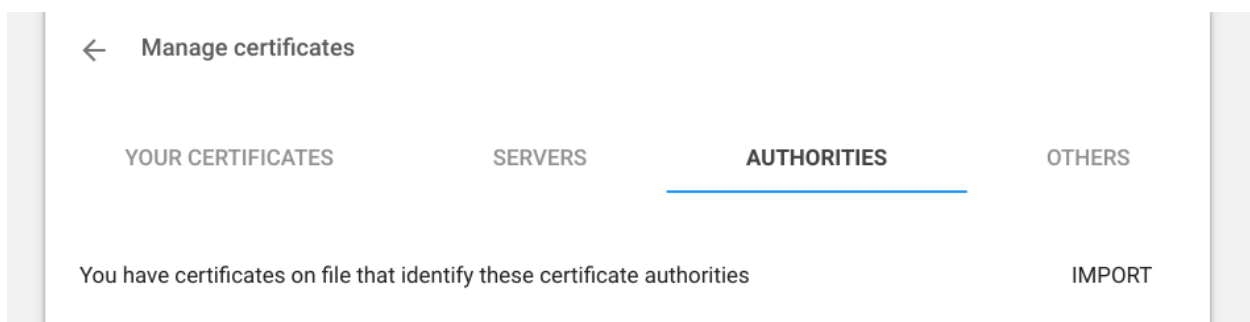
Open Chrome settings, scroll down to the very bottom and click on **Advanced** to expand the advanced settings.



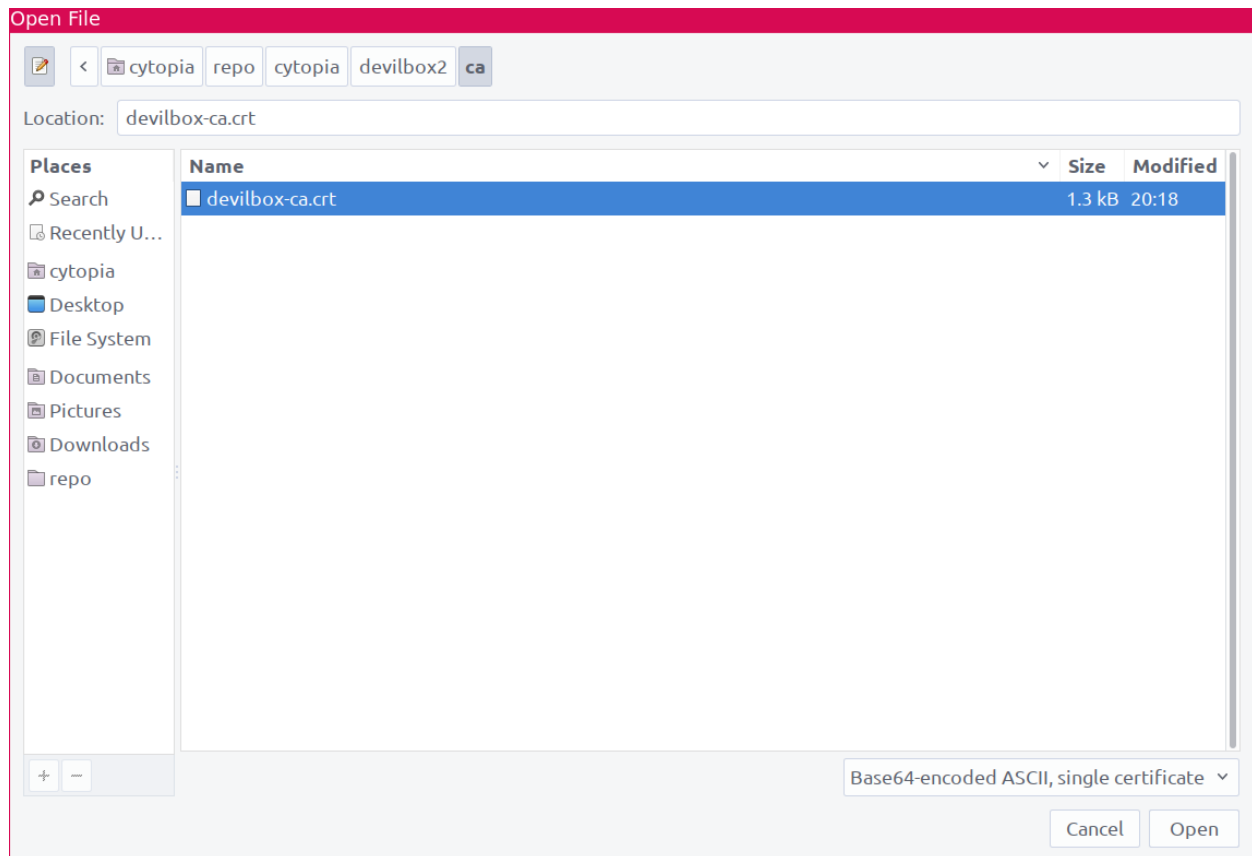
Find the setting `Manage certificates` and open it.



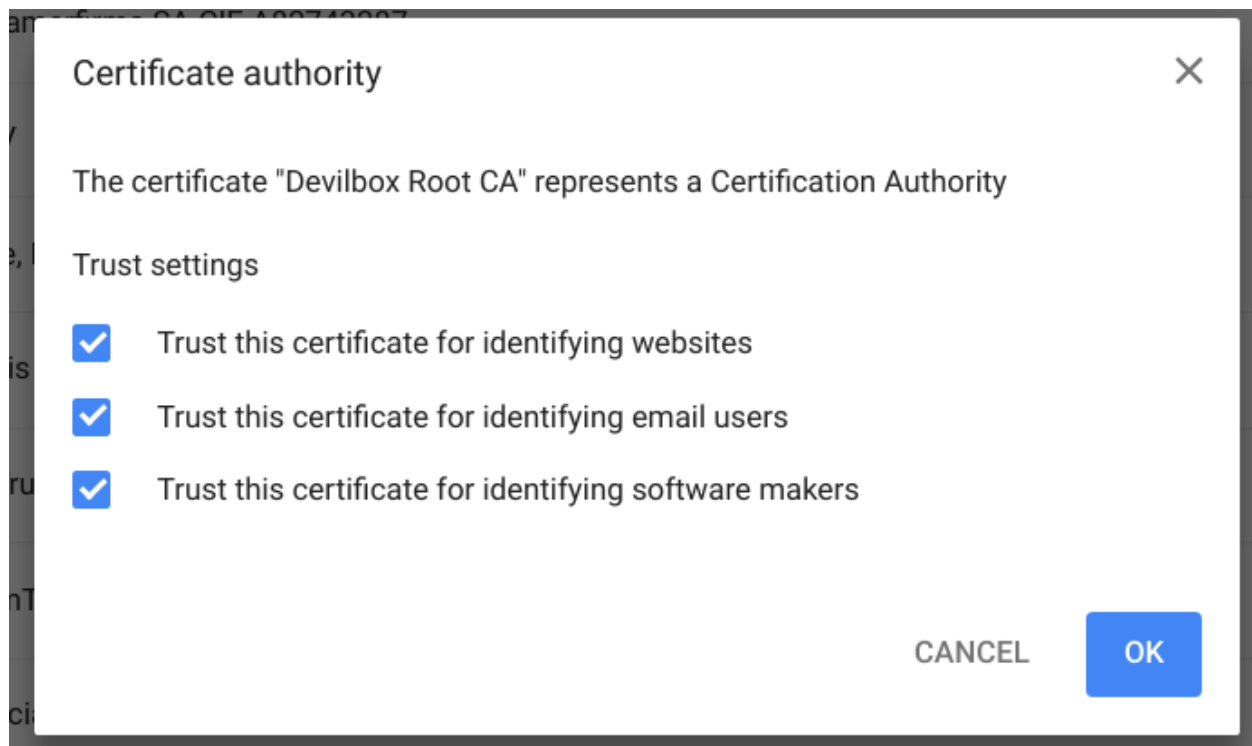
Navigate to the tab setting **AUTHORITIES** and click on **IMPORT**.



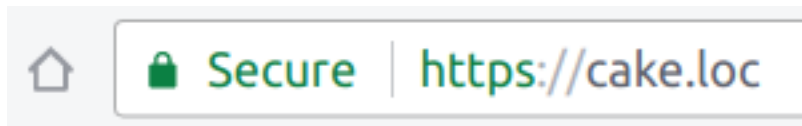
Select `devilbox-ca.crt` from within the Devilbox `./ca` directory:



As the last step you are asked what permissions you want to grant the newly importat CA. To make sure it works everywhere, check all options and proceed with OK.

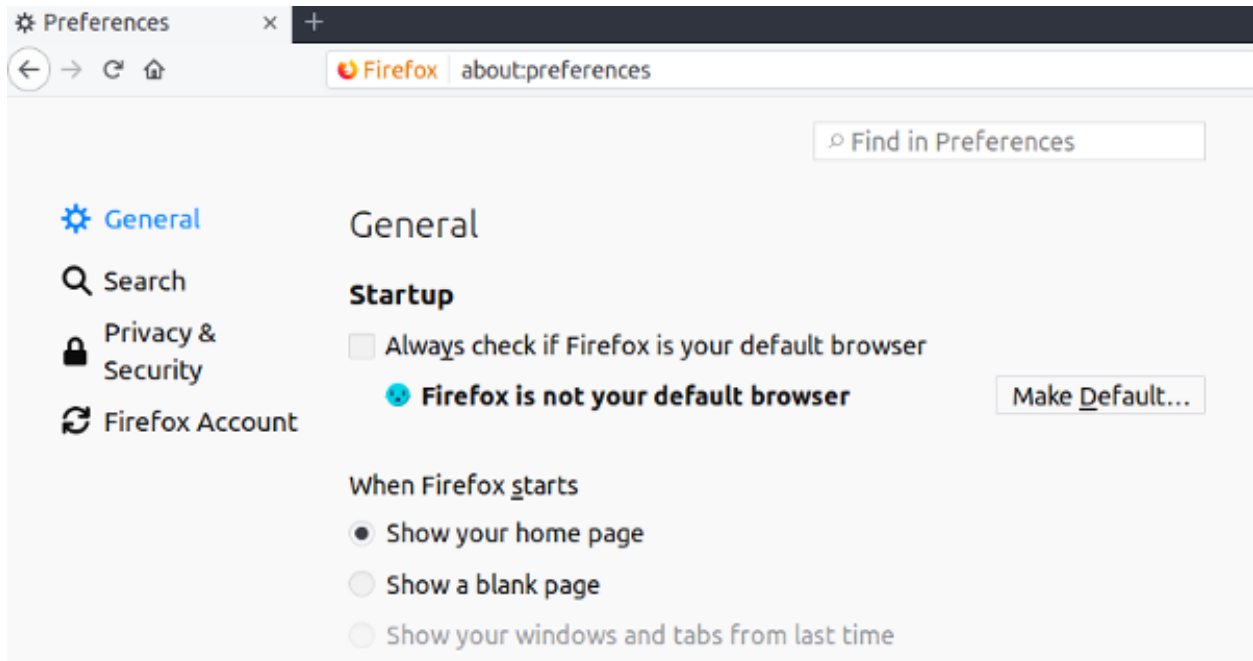


Now you are all set and all generated SSL certificates will be valid from now on.

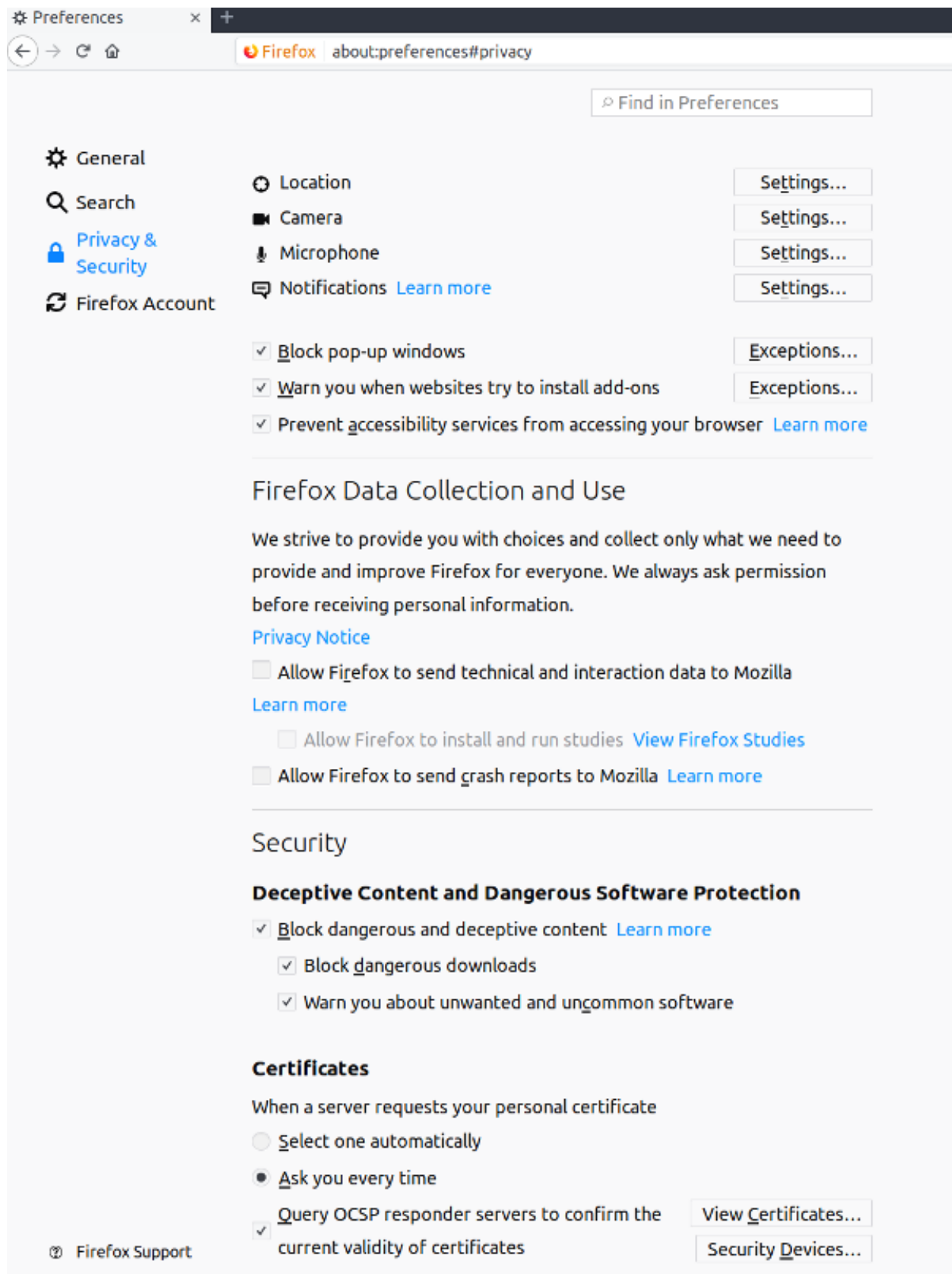


36.3.2 Firefox

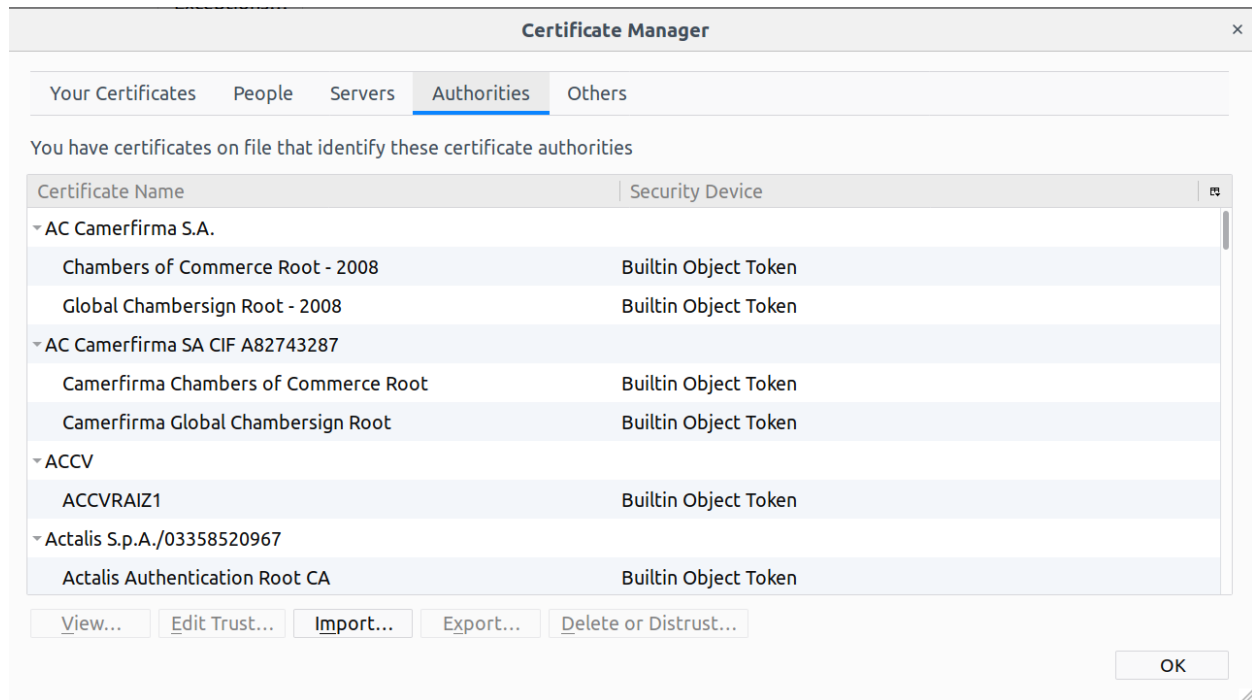
Open Firefox settings and click on Privacy & Security.



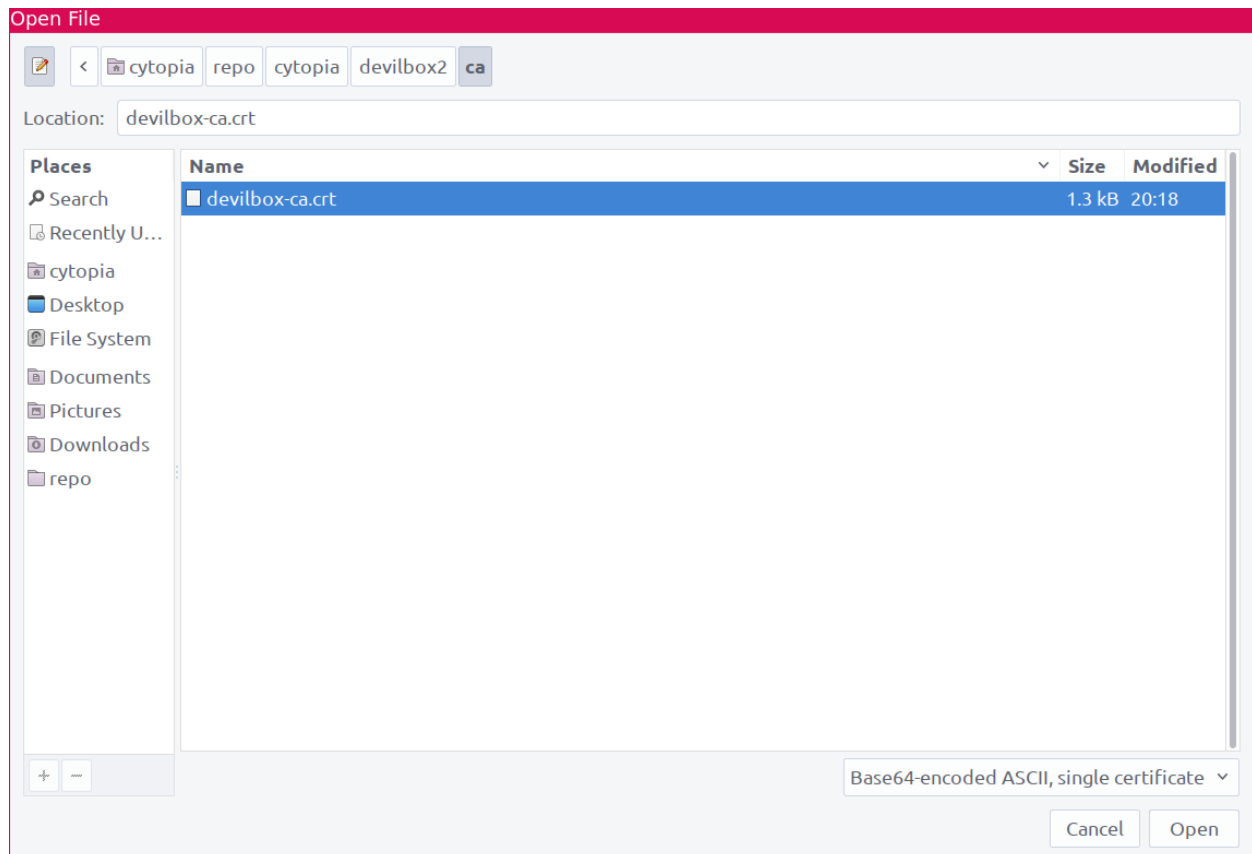
At the very bottom click on the button View Certificates.



In the Authorities tab, click on Import.



Select `devilbox-ca.crt` from within the Devilbox `./ca` directory:



As the last step you are asked what permissions you want to grant the newly importat CA. To make sure it works everywhere, check all options and proceed with OK.

Downloading Certificate

You have been asked to trust a new Certificate Authority (CA).

Do you want to trust "Devilbox Root CA" for the following purposes?

☒ Trust this CA to identify websites.

☒ Trust this CA to identify email users.

Before trusting this CA for any purpose, you should examine its certificate and its policy and procedures (if available).

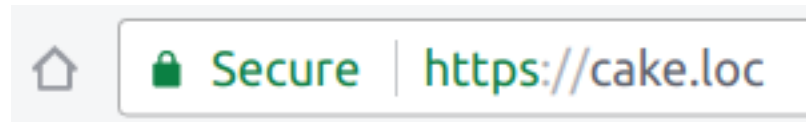
View

Examine CA certificate

Cancel

OK

Now you are all set and all generated SSL certificates will be valid from now on.



36.4 Further Reading

See also:

.env variable: *DEVILBOX_UI_SSL_CN*

This page lists a general overview about the bundled web server - its features, where it comes from, how it is built and what configuration is possible.

Table of Contents

- *Features*
 - *Auto-virtual hosts*
 - *File permission problem*
 - *Custom global configuration*
 - *Custom vhost configuration*
- *Information*
 - *Dockerfile*
 - *Github*
 - *Dockerhub*
 - *Build process*
- *Configuration*
 - *.env file*
 - *apache.conf / nginx.conf*
 - *vhost-gen*

37.1 Features

37.1.1 Auto-virtual hosts

37.1.2 File permission problem

37.1.3 Custom global configuration

37.1.4 Custom vhost configuration

37.2 Information

37.2.1 Dockerfile

37.2.2 Github

37.2.3 Dockerhub

37.2.4 Build process

37.3 Configuration

37.3.1 .env file

37.3.2 apache.conf / nginx.conf

37.3.3 vhost-gen

CHAPTER 38

PHP

Environment variables

CHAPTER 39

MySQL

CHAPTER 40

MongoDB

CHAPTER 41

Redis

CHAPTER 42

Memcached

CHAPTER 43

BIND

CHAPTER 44

Devilbox Intranet

If you don't want to add DNS records manually for every project, you can also use the bundled DNS server and use its DNS catch-all feature to have all DNS records automatically available.

Important: By default, the DNS server is set to listen on 1053 to avoid port collisions during startup. You need to change it to 53 in `.env` via `HOST_PORT_BIND`.

Table of Contents

- *Native Docker*
 - *Prerequisites*
 - *Linux*
 - *MacOS*
 - *Windows*
- *Docker Toolbox*
 - *MacOS*
 - *Windows*

45.1 Native Docker

The webserver as well as the DNS server must be available on `127.0.0.1` or on all interfaces on `0.0.0.0`. Additionally the DNS server port must be set to 53 (it is not by default).

- Ensure `LOCAL_LISTEN_ADDR` is set accordingly
- Ensure `HOST_PORT_BIND` is set accordingly

- No other DNS resolver should listen on 127.0.0.1:53

45.1.1 Prerequisites

First ensure that `LOCAL_LISTEN_ADDR` is either empty or listening on 127.0.0.1.

Listing 1: .env

```
host> cd path/to/devilbox
host> vi .env
LOCAL_LISTEN_ADDR=
```

Then you need to ensure that `HOST_PORT_BIND` is set to 53.

Listing 2: .env

```
host> cd path/to/devilbox
host> vi .env
HOST_PORT_BIND=53
```

Before starting up the Devilbox, ensure that port 53 is not already used on 127.0.0.1.

```
host> netstat -an | grep -E 'LISTEN\s*$'
tcp        0      0 127.0.0.1:53          0.0.0.0:*             LISTEN
tcp        0      0 127.0.0.1:43477       0.0.0.0:*             LISTEN
tcp        0      0 127.0.0.1:50267       0.0.0.0:*             LISTEN
```

If you see port 53 already being used as in the above example, ensure to stop any DNS resolver, otherwise it does not work.

The output should look like this (It is only important that there is no :53).

```
host> netstat -an | grep -E 'LISTEN\s*$'
tcp        0      0 127.0.0.1:43477       0.0.0.0:*             LISTEN
tcp        0      0 127.0.0.1:50267       0.0.0.0:*             LISTEN
```

45.1.2 Linux

If the prerequisites are met, you can edit `/etc/dhcp/dhclient.conf` with root or sudo privileges and add an instruction, which tells your local DHCP client that whenever any of your DNS servers are changed, you always want to have an additional entry, which is the one from the Devilbox.

Add the following line to to the very beginning of `/etc/dhcp/dhclient.conf`:

Listing 3: /etc/dhcp/dhclient.conf

```
prepend domain-name-servers 127.0.0.1;
```

When you do that for the first time, you need to restart the network-manager service.

```
# Via service command
$ sudo service network-manager restart

# Or the systemd way
$ sudo systemctl restart network-manager
```

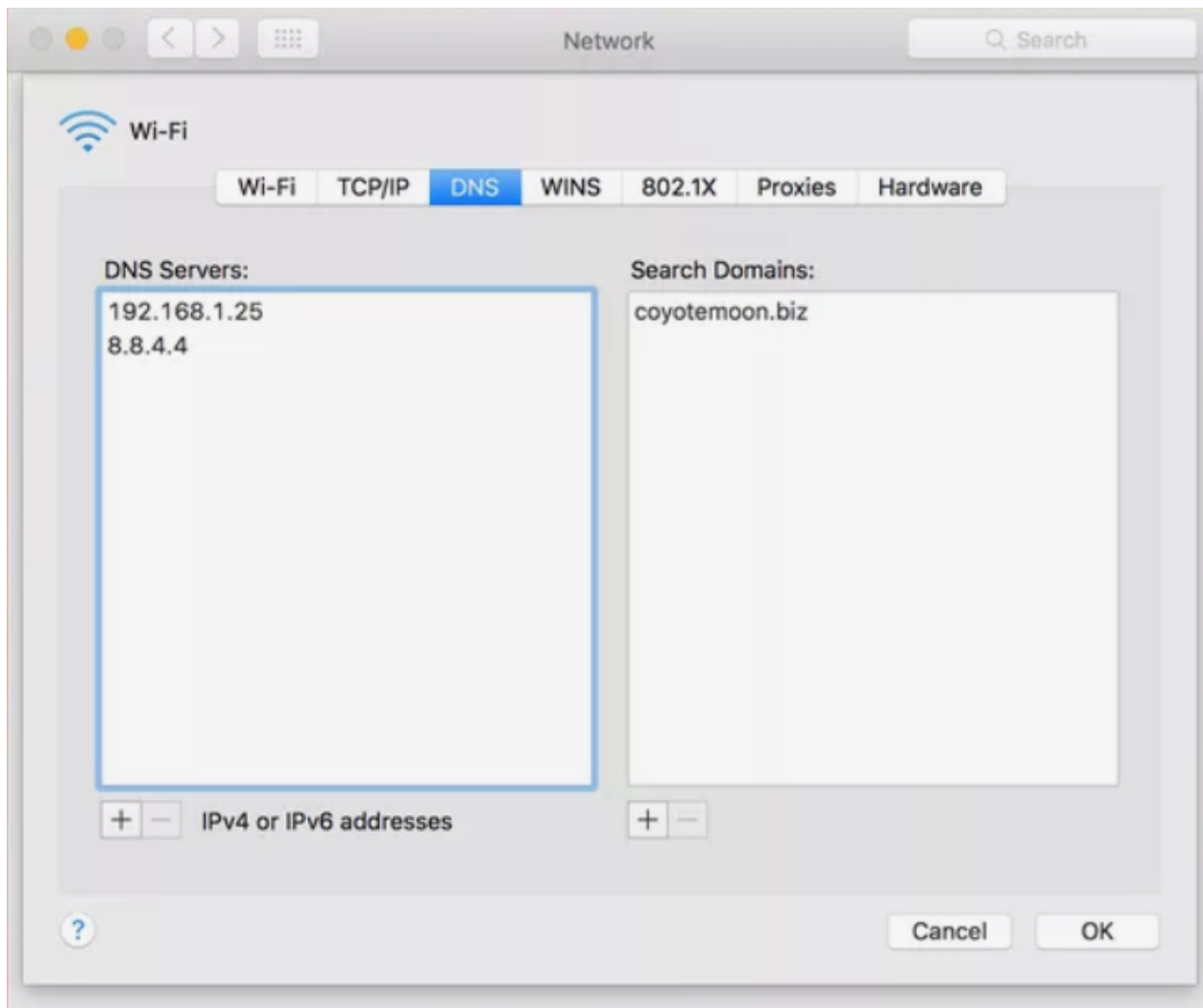
This will make sure that whenever your `/etc/resolv.conf` is deployed, you will have `127.0.0.1` as the first entry and also make use of any other DNS server which are deployed via the LAN's DHCP server.

If the Devilbox DNS server is not running, it does not affect the name resolution, because you will still have other entries in `/etc/resolv.conf`.

45.1.3 MacOS

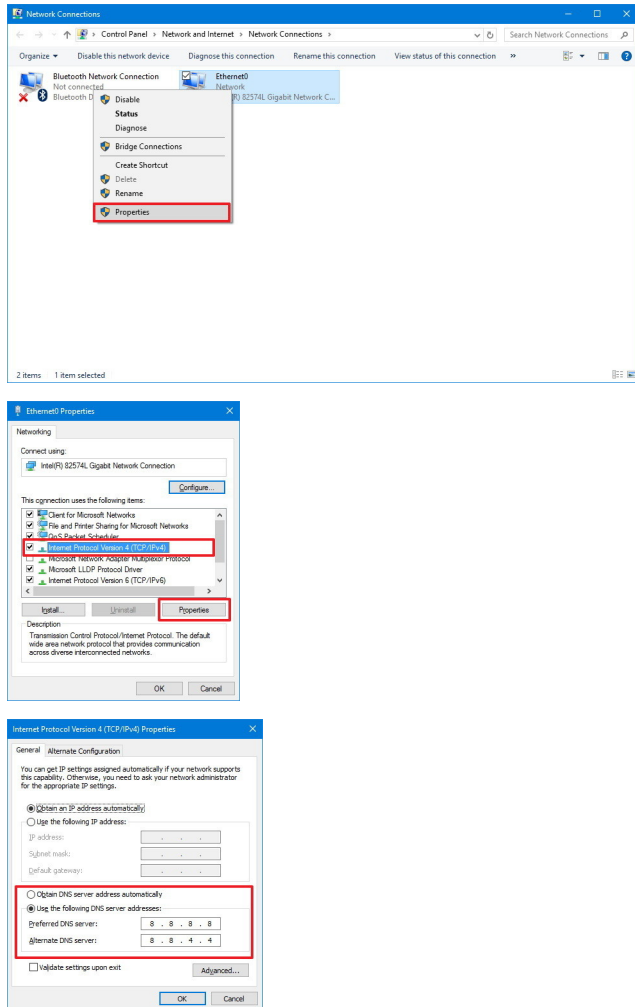
Modifying `/etc/resolv.conf` does not work on MacOS, you need to make changes in your System Preferences:

1. Open System Preferences
2. Go to Network
3. Select your connected interface
4. Click on DNS tab
5. Add new DNS server by clicking the + sign
6. Add `127.0.0.1`



45.1.4 Windows

On Windows, you need to change your active network adapter. See the following screenshots for how to do it.



In the last screenshot, you will have to add `127.0.0.1` as your Preferred DNS server.

45.2 Docker Toolbox

See also:

Docker Toolbox

45.2.1 MacOS

- `LOCAL_LISTEN_ADDR` must be empty in order to listen on all interfaces
- `HOST_PORT_BIND` must be set to 53
- Port 80 from the Docker Toolbox virtual machine must be port-forwarded to `127.0.0.1:80` on your host os
- Port 53 from the Docker Toolbox virtual machine must be port-forwarded to `127.0.0.1:53` on your host os

Todo: This section needs further proof and information.

45.2.2 Windows

- *LOCAL_LISTEN_ADDR* must be empty in order to listen on all interfaces
- *HOST_PORT_BIND* must be set to 53
- Port 80 from the Docker Toolbox virtual machine must be port-forwarded to 127.0.0.1:80 on your host os
- Port 53 from the Docker Toolbox virtual machine must be port-forwarded to 127.0.0.1:53 on your host os

Todo: This section needs further proof and information.

CHAPTER 46

.env file

All docker-compose configuration is done inside the `.env` file which simply defines key-value variables parsed to `docker-compose.yml`.

Note: what is the `.env` file?

Note: Use your browsers search function to quickly find the desired variable name.

Important: Any change of `.env` requires a restart of the Devilbox.

Table of Contents

- *Core settings*
 - *DEBUG_COMPOSE_ENTRYPOINT*
 - *DOCKER_LOGS*
 - *DEVILBOX_PATH*
 - *LOCAL_LISTEN_ADDR*
 - *TLD_SUFFIX*
 - *EXTRA_HOSTS*
 - *NEW_UID*
 - *NEW_GID*
 - *TIMEZONE*

- *Intranet settings*
 - *DNS_CHECK_TIMEOUT*
 - *DEVILBOX_UI_SSL_CN*
 - *DEVILBOX_UI_PROTECT*
 - *DEVILBOX_UI_PASSWORD*
 - *DEVILBOX_UI_ENABLE*
- *Docker image versions*
 - *PHP_SERVER*
 - *HTTPD_SERVER*
 - *MYSQL_SERVER*
 - *PGSQL_SERVER*
 - *REDIS_SERVER*
 - *MEMCD_SERVER*
 - *MONGO_SERVER*
- *Docker host mounts*
 - *HOST_PATH_HTTPD_DATADIR*
 - * *Example*
 - * *Mapping*
 - *HOST_PATH_MYSQL_DATADIR*
 - *HOST_PATH_PGSQL_DATADIR*
 - *HOST_PATH_MONGO_DATADIR*
- *Docker host ports*
 - *HOST_PORT_HTTPD*
 - *HOST_PORT_HTTPD_SSL*
 - *HOST_PORT_MYSQL*
 - *HOST_PORT_PGSQL*
 - *HOST_PORT_REDIS*
 - *HOST_PORT_MEMCD*
 - *HOST_PORT_MONGO*
 - *HOST_PORT_BIND*
- *Container settings*
 - *PHP*
 - * *Custom variables*
 - *Web server*
 - * *HTTPD_DOCROOT_DIR*

- * *HTTPD_TEMPLATE_DIR*
- *MySQL*
 - * *MYSQL_ROOT_PASSWORD*
 - * *MYSQL_GENERAL_LOG*
- *PostgreSQL*
 - * *PGSQL_ROOT_USER*
 - * *PGSQL_ROOT_PASSWORD*
- *Bind*
 - * *BIND_DNS_RESOLVER*
 - * *BIND_DNSSEC_VALIDATE*
 - * *BIND_LOG_DNS*
 - * *BIND_TTL_TIME*
 - * *BIND_REFRESH_TIME*
 - * *BIND_RETRY_TIME*
 - * *BIND_EXPIRY_TIME*
 - * *BIND_MAX_CACHE_TIME*

46.1 Core settings

46.1.1 DEBUG_COMPOSE_ENTRYPOINT

This variable controls the docker-compose log verbosity during service startup. When set to 1 verbose output as well as executed commands are shown. When set to 0 only warnings and errors are shown.

| Name | Allowed values | Default value |
|--------------------------|----------------|---------------|
| DEBUG_COMPOSE_ENTRYPOINT | 0 or 1 | 1 |

46.1.2 DOCKER_LOGS

This variable controls the output of logs. Logs can either go to file and will be available under `./log/` inside the Devilbox git directory or they can be forwarded to Docker logs and will then be send to stdout and stderr.

| Name | Allowed values | Default value |
|-------------|----------------|---------------|
| DOCKER_LOGS | 1 or 0 | 0 |

When `DOCKER_LOGS` is set to 1, output will go to Docker logs, otherwise if it is set to 0 the log output will go to files under `./log/`.

The `./log/` directory itself will contain subdirectories in the form `<service>-<version>` which will then hold all available log files.

Note: Log directories do not exist until you start the Devilbox and will only be created for the service versions you have enabled in `.env`.

The log directory structure would look something like this:

```
host> cd path/to/devilbox
host> tree log

log/
├── nginx-stable/
│   ├── nginx-stable/
│   ├── defaultlocalhost-access.log
│   ├── defaultlocalhost-error.log
│   ├── <project-name>-access.log    # Each project has its own access log
│   └── <project-name>-error.log    # Each project has its own error log
├── mariadb-10.1/
│   ├── error.log
│   ├── query.log
│   └── slow.log
└── php-fpm-7.1/
    ├── php-fpm.access
    └── php-fpm.error
```

When you want to read logs sent to Docker logs, you can do so via the following command:

```
host> cd path/to/devilbox
host> docker-compose logs
```

When you want to continuously watch the log output (such as `tail -f`), you need to append `-f` to the command.

```
host> cd path/to/devilbox
host> docker-compose logs -f
```

When you only want to have logs displayed for a single service, you can also append the service name (works with or without `-f` as well):

```
host> cd path/to/devilbox
host> docker-compose logs php -f
```

Important: Currently this is only implemented for PHP-FPM and HTTPD Docker container. MySQL will always output its logs to file and all other official Docker container always output to Docker logs.

46.1.3 DEVILBOX_PATH

This specifies a relative or absolute path to the Devilbox git directory and will be used as a prefix for all Docker mount paths.

- Relative path: relative to the devilbox git directory (Must start with `.`)
- Absolute path: Full path (Must start with `/`)

The only reason you would ever want change this variable is when you are on MacOS and relocate your project files onto an NFS volume due to performance issues.

Warning:

Remove stopped container Whenever you change this value you have to stop the Devilbox and also remove the stopped container via `docker-compose rm`.

| Name | Allowed values | Default value |
|---------------|----------------|---------------|
| DEVILBOX_PATH | valid path | . |

46.1.4 LOCAL_LISTEN_ADDR

This variable specifies you host computers listening IP address for exposed container ports. If you leave this variable empty, all exposed ports will be bound to all network interfaces on your host operating system, which is also the default behaviour. If you only want the exposed container ports to be bound to a specific IP address (such as `127.0.0.1`), you can add this IP address here, but note, in this case you must add a trailing colon (`:`).

| Name | Allowed values | Default value |
|-------------------|----------------|---------------|
| LOCAL_LISTEN_ADDR | IP address | empty |

Examples:

| Value | Meaning |
|---------------------------|--|
| <code>127.0.0.1:</code> | only expose ports on your host os on <code>127.0.0.1</code> . Note the trailing <code>:</code> |
| <code>192.168.0.1:</code> | only expose ports on your host os on <code>192.168.0.1</code> . Note the trailing <code>:</code> |
| <code>0.0.0.0:</code> | listen on all host computer interfaces / IP addresses |
| | listen on all host computer interfaces / IP addresses |

Note: When using `Docker Toolbox`, you must leave this variable empty, in order to have the exposed ports available on the external interface of the virtual machine.

46.1.5 TLD_SUFFIX

This variable controls all of your projects domain suffix.

| Name | Allowed values | Default value |
|------------|------------------|------------------|
| TLD_SUFFIX | alpha-num string | <code>loc</code> |

Your project domains are built together out of the project directory name and the `TLD_SUFFIX`. The formula is like this: `http://<project-dir>.<TLD_SUFFIX>`.

You can even use official tld's and have your nameserver point to an internal LAN id, to make this project visible to everyone in your corporate LAN.

How does it look?

| Project dir | TLD_SUFFIX | Project URL |
|-------------|------------|-----------------------|
| my-test | loc | http://my-test.loc |
| example | loc | http://example.loc |
| www.test | loc | http://www.test.loc |
| my-test | local | http://my-test.local |
| example | local | http://example.local |
| www.test | local | http://www.test.local |
| my-test | net | http://my-test.net |
| example | com | http://example.com |
| www.test | org | http://www.test.org |

Warning: Do not use `dev` as a domain suffix (I know, it's tempting). It has been registered by [Google](#) and they advertise the [HSTS header](#) which makes your browser redirect every http request to https.

See also: [This blog post](#)

Warning: Do not use `localhost` as a domain suffix. There is an RFC draft to make sure all localhost requests, including their sub domains should be redirected to the systems loopback interface. Docker has already released a commit preventing the use of `localhost` on MacOS.

See also: [RFC Draft](#) and [Docker Release notes](#)

46.1.6 EXTRA_HOSTS

This variable allows you to add additional DNS entries from hosts outside the Devilbox network, such as hosts running on your host operating system, the LAN or from the internet.

| Name | Allowed values | Default value |
|-------------|------------------------------|---------------|
| EXTRA_HOSTS | comma separated host mapping | empty |

Adding hosts can be done in two ways:

1. Add DNS entry for an IP address
2. Add DNS entry for a hostname/CNAME which will be mapped to whatever IP address it will resolve

The general structure to add extra hosts looks like this

```
# Single host
EXTRA_HOSTS='hostname=1.1.1.1'
EXTRA_HOSTS='hostname=CNAME'

# Multiple hosts
EXTRA_HOSTS='hostname1=1.1.1.1,hostname2=2.2.2.2'
EXTRA_HOSTS='hostname1=CNAME1,hostname2=CNAME2'
```

- The left side represents the name by which the host will be available by
- The right side represents the IP address by which the new name will resolve to
- If the right side is a CNAME itself, it will be first resolved to an IP address and then the left side will resolve to that IP address.

A few examples for adding extra hosts:

```
# 1. One entry:
# The following extra host 'loc' is added and will always point to 192.168.0.7.
# When reverse resolving '192.168.0.7' it will answer with 'tld'.
EXTRA_HOSTS='loc=192.168.0.7'

# 2. One entry:
# The following extra host 'my.host.loc' is added and will always point to 192.168.0.
→ 9.
# When reverse resolving '192.168.0.9' it will answer with 'my.host'.
EXTRA_HOSTS='my.host.loc=192.168.0.9'

# 3. Two entries:
# The following extra host 'tld' is added and will always point to 192.168.0.1.
# When reverse resolving '192.168.0.1' it will answer with 'tld'.
# A second extra host 'example.org' is added and always redirects to 192.168.0.2
# When reverse resolving '192.168.0.2' it will answer with 'example.org'.
EXTRA_HOSTS='tld=192.168.0.1,example.org=192.168.0.2'

# 4. Using CNAME's for resolving:
# The following extra host 'my.host' is added and will always point to whatever
# IP example.org resolves to.
# When reverse resolving '192.168.0.1' it will answer with 'my.host'.
EXTRA_HOSTS='my.host=example.org'
```

See also:

This resembles the feature of Docker Compose: `extra_hosts` to add external links.

See also:

Communicating with external hosts

46.1.7 NEW_UID

This setting controls one of the core concepts of the Devilbox. It overcomes the problem of synchronizing file and directory permissions between the Docker container and your host operating system.

You should set this value to the user id of your host operating systems user you actually work with. How do you find out your user id?

```
host> id -u
1000
```

In most cases (on Linux and MacOS), this will be 1000 if you are the first and only user on your system, however it could also be a different value.

| Name | Allowed values | Default value |
|---------|----------------|---------------|
| NEW_UID | valid uid | 1000 |

The Devilbox own containers will then pick up this value during startup and change their internal user id to the one specified. Services like PHP-FPM, Apache and Nginx will then do read and write operation of files with this uid, so all files mounted will have permissions as your local user and you do not have to fix permissions afterwards.

See also:

Synchronize container permissions Read up more on the general problem of trying to have synchronized permissions between the host system and a running Docker container.

46.1.8 NEW_GID

This is the equivalent to user id for groups and addresses the same concept. See *NEW_UID*.

How do you find out your group id?

```
host> id -g
1000
```

In most cases (on Linux and MacOS), this will be 1000 if you are the first and only user on your system, however it could also be a different value.

| Name | Allowed values | Default value |
|---------|----------------|---------------|
| NEW_GID | valid gid | 1000 |

See also:

Synchronize container permissions Read up more on the general problem of trying to have synchronized permissions between the host system and a running Docker container.

46.1.9 TIMEZONE

This variable controls the system as well as service timezone for the Devilbox's own containers. This is especially useful to keep PHP and database timezones in sync.

| Name | Allowed values | Default value |
|----------|----------------|---------------|
| TIMEZONE | valid timezone | Europe/Berlin |

Have a look at Wikipedia to get a list of valid timezones: https://en.wikipedia.org/wiki/List_of_tz_database_time_zones

Note: It is always a good practice not to assume a specific timezone anyway and store all values in UTC (such as time types in MySQL).

46.2 Intranet settings

46.2.1 DNS_CHECK_TIMEOUT

The Devilbox intranet validates if every project has a corresponding DNS record (either an official DNS record, one that came from its own Auto-DNS or an `/etc/hosts` entry). By doing so it queries the DNS record based on `<project-dir>.<TLD_SUFFIX>`. In case it does not exist, the query itself might take a while and the intranet page will be unresponsive during that time. In order to avoid long waiting times, you can set the DNS query time-out in seconds after which the query should stop and report as unsuccessful. The default is 1 second, wich should be fairly sane for all use-cases.

| Name | Allowed values | Default value |
|-------------------|----------------|---------------|
| DNS_CHECK_TIMEOUT | integers | 1 |

46.2.2 DEVILBOX_UI_SSL_CN

When accessing the Devilbox intranet via `https` it will use an automatically created SSL certificate. Each SSL certificate requires a valid Common Name, which must match the virtual host name.

This setting let's you specify by what **name** you are accessing the Devilbox intranet. The default is `localhost`, but if you have created your own alias, you must change this value accordingly. Also note that multiple values are possible and must be separated with a comma. When you add an asterisk (`*`) to the beginning, it means it will create a wildcard certificate for that hostname.

| Name | Allowed values | Default value |
|--------------------|------------------------------|---|
| DEVILBOX_UI_SSL_CN | comma separated list of CN's | <code>localhost, *.localhost, devilbox, *.devilbox</code> |

Examples:

- `DEVILBOX_UI_SSL_CN=localhost`
- `DEVILBOX_UI_SSL_CN=localhost, *.localhost`
- `DEVILBOX_UI_SSL_CN=localhost, *.localhost, devilbox, *.devilbox`
- `DEVILBOX_UI_SSL_CN=intranet.example.com`

See also:

[HTTPS \(SSL\)](#)

46.2.3 DEVILBOX_UI_PROTECT

By setting this variable to 1, the Devilbox intranet will be password protected. This might be useful, if you share your running Devilbox instance accross a LAN, but do not want everybody to have access to the intranet itself, just to the projects you actually provide.

| Name | Allowed values | Default value |
|---------------------|----------------|---------------|
| DEVILBOX_UI_PROTECT | 0 or 1 | 0 |

Note: Also pay attention to the next env var, which will control the password for the login: `DEVILBOX_UI_PASSWORD`.

46.2.4 DEVILBOX_UI_PASSWORD

When the devilbox intranet is password-protected via `DEVILBOX_UI_PROTECT`, this is the actual password by which it will be protected.

| Name | Allowed values | Default value |
|----------------------|----------------|-----------------------|
| DEVILBOX_UI_PASSWORD | any string | <code>password</code> |

46.2.5 DEVILBOX_UI_ENABLE

In case you want to completely disable the Devilbox intranet, such as when running it on production, you need to set this variable to 0.

By disabling the intranet, the webserver will simply remove the default virtual host and redirect all IP-based requests to the first available virtual host, which will be your first project when ordering their names alphabetically.

| Name | Allowed values | Default value |
|--------------------|----------------|---------------|
| DEVILBOX_UI_ENABLE | 0 or 1 | 1 |

46.3 Docker image versions

The following settings reflect one of the main goals of the Devilbox: being able to run any combination of all container versions.

Note: Any change for those settings requires a restart of the devilbox.

46.3.1 PHP_SERVER

This variable chooses your desired PHP-FPM version to be started.

| Name | Allowed values | Default value |
|------------|--|---------------|
| PHP_SERVER | php-fpm-5.4 php-fpm-5.5 php-fpm-5.6 php-fpm-7.0 php-fpm-7.1 php-fpm-7.2 | php-fpm-7.1 |

All values are already available in the `.env` file and just need to be commented or uncommented. If multiple values are uncommented, the last uncommented variable one takes precedence:

Listing 1: `.env`

```
host> grep PHP_SERVER .env

#PHP_SERVER=php-fpm-5.4
#PHP_SERVER=php-fpm-5.5
#PHP_SERVER=php-fpm-5.6
#PHP_SERVER=php-fpm-7.0
PHP_SERVER=php-fpm-7.1
#PHP_SERVER=php-fpm-7.2
#PHP_SERVER=php-fpm-7.3
#PHP_SERVER=hhvm-latest
```

46.3.2 HTTPD_SERVER

This variable chooses your desired web server version to be started.

| Name | Allowed values | Default value |
|--------------|--|---------------|
| HTTPD_SERVER | apache-2.2 apache-2.4 nginx-stable nginx-mainline | nginx-stable |

All values are already available in the `.env` file and just need to be commented or uncommented. If multiple values are uncommented, the last uncommented variable one takes precedences:

Listing 2: `.env`

```
host> grep HTTPD_SERVER .env

#HTTPD_SERVER=apache-2.2
#HTTPD_SERVER=apache-2.4
HTTPD_SERVER=nginx-stable
#HTTPD_SERVER=nginx-mainline
```

46.3.3 MYSQL_SERVER

This variable choses your desired MySQL server version to be started.

| Name | Allowed values | Default value |
|--------------|--|---------------|
| MYSQL_SERVER | mysql-5.5 mysql-5.6 mariadb-10.2 percona-5.7 and many more | mariadb-10.1 |

All values are already available in the `.env` file and just need to be commented or uncommented. If multiple values are uncommented, the last uncommented variable one takes precedences:

Listing 3: `.env`

```
host> grep MYSQL_SERVER .env

#MYSQL_SERVER=mysql-5.5
#MYSQL_SERVER=mysql-5.6
#MYSQL_SERVER=mysql-5.7
#MYSQL_SERVER=mysql-8.0
#MYSQL_SERVER=mariadb-5.5
#MYSQL_SERVER=mariadb-10.0
MYSQL_SERVER=mariadb-10.1
#MYSQL_SERVER=mariadb-10.2
#MYSQL_SERVER=mariadb-10.3
#MYSQL_SERVER=percona-5.5
#MYSQL_SERVER=percona-5.6
#MYSQL_SERVER=percona-5.7
```

46.3.4 PGSQL_SERVER

This variable choses your desired PostgreSQL server version to be started.

| Name | Allowed values | Default value |
|--------------|-------------------------------|---------------|
| PGSQL_SERVER | 9.1 9.2 9.3 9.4 and many more | 9.6 |

All values are already available in the `.env` file and just need to be commented or uncommented. If multiple values are uncommented, the last uncommented variable one takes precedences:

Listing 4: .env

```
host> grep PGSQL_SERVER .env

#PGSQL_SERVER=9.1
#PGSQL_SERVER=9.2
#PGSQL_SERVER=9.3
#PGSQL_SERVER=9.4
#PGSQL_SERVER=9.5
PGSQL_SERVER=9.6
#PGSQL_SERVER=10.0
```

Note: This is the official PostgreSQL server which might already have other tags available, check their official website for even more versions. https://hub.docker.com/_/postgres/

46.3.5 REDIS_SERVER

This variable choses your desired Redis server version to be started.

| Name | Allowed values | Default value |
|--------------|-------------------------------|---------------|
| REDIS_SERVER | 2.8 3.0 3.2 4.0 and many more | 4.0 |

All values are already available in the .env file and just need to be commented or uncommented. If multiple values are uncommented, the last uncommented variable one takes precedences:

Listing 5: .env

```
host> grep REDIS_SERVER .env

#REDIS_SERVER=2.8
#REDIS_SERVER=3.0
#REDIS_SERVER=3.2
REDIS_SERVER=4.0
```

Note: This is the official Redis server which might already have other tags available, check their official website for even more versions. https://hub.docker.com/_/redis/

46.3.6 MEMCD_SERVER

This variable choses your desired Memcached server version to be started.

| Name | Allowed values | Default value |
|--------------|---|---------------|
| MEMCD_SERVER | 1.4.21 1.4.22 1.4.23 1.4.24 and many more | 1.5.2 |

All values are already available in the .env file and just need to be commented or uncommented. If multiple values are uncommented, the last uncommented variable one takes precedences:

Listing 6: .env

```

host> grep MEMCD_SERVER .env

#MEMCD_SERVER=1.4.21
#MEMCD_SERVER=1.4.22
#MEMCD_SERVER=1.4.23
#MEMCD_SERVER=1.4.24
#MEMCD_SERVER=1.4.25
#MEMCD_SERVER=1.4.26
#MEMCD_SERVER=1.4.27
#MEMCD_SERVER=1.4.28
#MEMCD_SERVER=1.4.29
#MEMCD_SERVER=1.4.30
#MEMCD_SERVER=1.4.31
#MEMCD_SERVER=1.4.32
#MEMCD_SERVER=1.4.33
#MEMCD_SERVER=1.4.34
#MEMCD_SERVER=1.4.35
#MEMCD_SERVER=1.4.36
#MEMCD_SERVER=1.4.37
#MEMCD_SERVER=1.4.38
#MEMCD_SERVER=1.4.39
#MEMCD_SERVER=1.5.0
#MEMCD_SERVER=1.5.1
MEMCD_SERVER=1.5.2
#MEMCD_SERVER=latest

```

Note: This is the official Memcached server which might already have other tags available, check their official website for even more versions. https://hub.docker.com/_/memcached/

46.3.7 MONGO_SERVER

This variable chooses your desired MongoDB server version to be started.

| Name | Allowed values | Default value |
|--------------|-------------------------------|---------------|
| MONGO_SERVER | 2.8 3.0 3.2 3.4 and many more | 3.4 |

All values are already available in the .env file and just need to be commented or uncommented. If multiple values are uncommented, the last uncommented variable one takes precedences:

Listing 7: .env

```

host> grep MONGO_SERVER .env

#MONGO_SERVER=2.8
#MONGO_SERVER=3.0
#MONGO_SERVER=3.2
MONGO_SERVER=3.4
#MONGO_SERVER=3.5

```

Note: This is the official MongoDB server which might already have other tags available, check their official website

for even more versions. https://hub.docker.com/_/mongo/

46.4 Docker host mounts

The Docker host mounts are directory paths on your host operating system that will be mounted into the running Docker container. This makes data persistent accross restarts and let them be available on both sides: Your host operating system as well as inside the container.

This also gives you the choice to edit data on your host operating system, such as with your favourite IDE/editor and also inside the container, by using the bundled tools, such as downloading libraries with `composer` and others.

Being able to do that on both sides, removes the need to install any development tools (except your IDE/editor) on your host and have everything fully encapsulated into the containers itself.

46.4.1 HOST_PATH_HTTPD_DATADIR

This is an absolute or relative path (relative to Devilbox git directory) to your data directory.

See also:

Data directory

By default, all of your websites/projects will be stored in that directory. If however you want to separate your data from the Devilbox git directory, do change the path to a place where you want to store all of your projects on your host computer.

- Relative path: relative to the devilbox git directory (Must start with `.`)
- Absolute path: Full path (Must start with `/`)

| Name | Allowed values | Default value |
|-------------------------|----------------|-------------------------|
| HOST_PATH_HTTPD_DATADIR | valid path | <code>./data/www</code> |

Example

If you want to move all your projects to `/home/myuser/workspace/web/` for example, just set it like this:

Listing 8: `.env`

```
HOST_PATH_HTTPD_DATADIR=/home/myuser/workspace/web
```

Mapping

No matter what path you assign, inside the PHP and the web server container your data dir will always be `/shared/httpd/`.

Warning: Do not create any symlinks inside your project directories that go outside the data dir. Anything which is outside this directory is not mounted into the container.

Warning:

Remove stopped container Whenever you change this value you have to stop the Devilbox and also remove the stopped container via `docker-compose rm`.

46.4.2 HOST_PATH_MYSQL_DATADIR

This is an absolute or relative path (relative to Devilbox git directory) to your MySQL data directory.

- Relative path: relative to the devilbox git directory (Must start with `.`)
- Absolute path: Full path (Must start with `/`)

| Name | Allowed values | Default value |
|-------------------------|----------------|---------------------------|
| HOST_PATH_MYSQL_DATADIR | valid path | <code>./data/mysql</code> |

Each MySQL, MariaDB or PerconaDB version will have its own subdirectory, so when first running MySQL 5.5 and then starting MySQL 5.6, you will have a different database with different data.

Having each version separated from each other makes sure that you don't accidentally upgrade from a lower to a higher version which might not be reversible. (MySQL auto-upgrade certain older data files to newer, but this process does not necessarily work the other way round and could result in failures).

The directory structure will look something like this:

```
host> ls -l ./data/mysql/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 mariadb-10.0/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 mariadb-10.1/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 mariadb-10.2/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 mariadb-10.3/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 mysql-5.5/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 mysql-5.6/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 mysql-5.7/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 mysql-8.0/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 percona-5.5/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 percona-5.6/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 percona-5.7/
```

Warning:

Remove stopped container Whenever you change this value you have to stop the Devilbox and also remove the stopped container via `docker-compose rm`.

46.4.3 HOST_PATH_PGSQL_DATADIR

This is an absolute or relative path (relative to Devilbox git directory) to your PostgreSQL data directory.

- Relative path: relative to the devilbox git directory (Must start with `.`)
- Absolute path: Full path (Must start with `/`)

| Name | Allowed values | Default value |
|-------------------------|----------------|---------------------------|
| HOST_PATH_PGSQL_DATADIR | valid path | <code>./data/pgsql</code> |

Each PostgreSQL version will have its own subdirectory, so when first running PostgreSQL 9.1 and then starting PostgreSQL 10.0, you will have a different database with different data.

Having each version separated from each other makes sure that you don't accidentally upgrade from a lower to a higher version which might not be reversible.

The directory structure will look something like this:

```
host> ls -l ./data/pgsql/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 9.1/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 9.2/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 9.3/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 9.4/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 9.5/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 9.6/
```

Warning:

Remove stopped container Whenever you change this value you have to stop the Devilbox and also remove the stopped container via `docker-compose rm`.

46.4.4 HOST_PATH_MONGO_DATADIR

This is an absolute or relative path (relative to Devilbox git directory) to your MongoDB data directory.

- Relative path: relative to the devilbox git directory (Must start with `.`)
- Absolute path: Full path (Must start with `/`)

| Name | Allowed values | Default value |
|-------------------------|----------------|---------------------------|
| HOST_PATH_MONGO_DATADIR | valid path | <code>./data/mongo</code> |

Each MongoDB version will have its own subdirectory, so when first running MongoDB 2.8 and then starting MongoDB 3.5, you will have a different database with different data.

Having each version separated from each other makes sure that you don't accidentally upgrade from a lower to a higher version which might not be reversible.

The directory structure will look something like this:

```
host> ls -l ./data/mongo/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 2.8/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 3.0/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 3.2/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 3.4/
drwxrwxr-x 6 48 48 4096 Jun 21 08:47 3.5/
```

Warning:

Remove stopped container Whenever you change this value you have to stop the Devilbox and also remove the stopped container via `docker-compose rm`.

46.5 Docker host ports

All described host ports below are ports that the Docker container expose on your host operating system. By default each port will be exposed to all interfaces or IP addresses of the host operating system. This can be controlled with [LOCAL_LISTEN_ADDR](#).

How to list used ports on Linux and MacOS

Open a terminal and type the following:

```
host> netstat -an | grep 'LISTEN\s'
tcp        0      0 127.0.0.1:53585      0.0.0.0:*            LISTEN
tcp        0      0 127.0.0.1:37715      0.0.0.0:*            LISTEN
tcp        0      0 127.0.0.1:58555      0.0.0.0:*            LISTEN
tcp        0      0 127.0.0.1:48573      0.0.0.0:*            LISTEN
tcp        0      0 127.0.0.1:34591      0.0.0.0:*            LISTEN
tcp        0      0 127.0.0.1:8000       0.0.0.0:*            LISTEN
```

How to list used ports on Windows

Open the command prompt and type the following:

```
C:\WINDOWS\system32> netstat -an
Proto Local Address          Foreign Address         State
TCP    0.0.0.0:80              0.0.0.0:0               LISTENING
TCP    0.0.0.0:145            0.0.0.0:0               LISTENING
TCP    0.0.0.0:445            0.0.0.0:0               LISTENING
TCP    0.0.0.0:1875           0.0.0.0:0               LISTENING
```

Warning:

Docker Toolbox When using Docker Toolbox ensure that ports are exposed to all interfaces. See [LOCAL_LISTEN_ADDR](#)

Warning: Before setting the ports, ensure that they are not already in use on your host operating system by other services.

46.5.1 HOST_PORT_HTTPD

The port to expose for the web server (Apache or Nginx). This is usually 80. Set it to something else if 80 is already in use on your host operating system.

| Name | Allowed values | Default value |
|-----------------|----------------|---------------|
| HOST_PORT_HTTPD | 1 - 65535 | 80 |

46.5.2 HOST_PORT_HTTPD_SSL

The port to expose for the web server (Apache or Nginx) for HTTPS (SSL) requests. This is usually 443. Set it to something else if 443 is already in use on your host operating system.

46.5.3 HOST_PORT_MYSQL

The port to expose for the MySQL server (MySQL, MariaDB or PerconaDB). This is usually 3306. Set it to something else if 3306 is already in use on your host operating system.

| Name | Allowed values | Default value |
|-----------------|----------------|---------------|
| HOST_PORT_MYSQL | 1 - 65535 | 3306 |

46.5.4 HOST_PORT_PGSQL

The port to expose for the PostgreSQL server. This is usually 5432. Set it to something else if 5432 is already in use on your host operating system.

| Name | Allowed values | Default value |
|-----------------|----------------|---------------|
| HOST_PORT_PGSQL | 1 - 65535 | 5432 |

46.5.5 HOST_PORT_REDIS

The port to expose for the Redis server. This is usually 6379. Set it to something else if 6379 is already in use on your host operating system.

| Name | Allowed values | Default value |
|-----------------|----------------|---------------|
| HOST_PORT_REDIS | 1 - 65535 | 6379 |

46.5.6 HOST_PORT_MEMCD

The port to expose for the Memcached server. This is usually 11211. Set it to something else if 11211 is already in use on your host operating system.

| Name | Allowed values | Default value |
|-----------------|----------------|---------------|
| HOST_PORT_MEMCD | 1 - 65535 | 11211 |

46.5.7 HOST_PORT_MONGO

The port to expose for the MongoDB server. This is usually 27017. Set it to something else if 27017 is already in use on your host operating system.

| Name | Allowed values | Default value |
|-----------------|----------------|---------------|
| HOST_PORT_MONGO | 1 - 65535 | 27017 |

46.5.8 HOST_PORT_BIND

The port to expose for the BIND DNS server. This is usually 53. Set it to something else if 53 is already in use on your host operating system.

| Name | Allowed values | Default value |
|----------------|----------------|---------------|
| HOST_PORT_BIND | 1 - 65535 | 53 |

Warning: As you might have noticed, BIND is not set to its default port 53 by default, but rather to 1053. This is because some operating system already have a local DNS resolver running on port 53 which would result in a failure when this BIND server is starting.

You only need to set BIND to port 53 when you want to use the `Auto-DNS` feature of the Devilbox. When doing so, read this article with care: [Auto-DNS](#).

46.6 Container settings

46.6.1 PHP

Custom variables

The PHP container itself does not offer any variables, however you can add any key-value pair variable into the `.env` file which will automatically be available to the started PHP container and thus in any of your PHP projects.

If your application requires a variable to determine if it is run under development or production, for example: `APPLICATION_ENV`, you can just add this to the `.env` file:

Listing 9: `.env`

```
host> grep APPLICATION_ENV .env
APPLICATION_ENV=development
```

Within your php application/file you can then access this variable via the `getenv` function:

Listing 10: `index.php`

```
<?php
// Example use of getenv()
echo getenv('APPLICATION_ENV');
?>
```

This will then output `development`.

Note: Add as many custom environment variables as you require.

See also:

[Custom environment variables](#)

46.6.2 Web server

`HTTPD_DOCROOT_DIR`

This variable specifies the name of a directory within each of your project directories from which the web server will serve the files.

Together with the `HOST_PATH_HTTPD_DATADIR` and your project directory, the `HTTPD_DOCROOT_DIR` will build up the final location of a virtual hosts document root.

| Name | Allowed values | Default value |
|-------------------|----------------|---------------|
| HTTPD_DOCROOT_DIR | valid dir name | htdocs |

Example 1

- devilbox git directory location: /home/user-1/repo/devilbox
- HOST_PATH_HTTPD_DATADIR: ./data/www (relative)
- Project directory: my-first-project
- HTTPD_DOCROOT_DIR: htdocs

The location from where the web server will serve files for my-first-project is then: /home/user-1/repo/devilbox/data/www/my-first-project/htdocs

Example 2

- devilbox git directory location: /home/user-1/repo/devilbox
- HOST_PATH_HTTPD_DATADIR: /home/user-1/www (absolute)
- Project directory: my-first-project
- HTTPD_DOCROOT_DIR: htdocs

The location from where the web server will serve files for my-first-project is then: /home/user-1/www/my-first-project/htdocs

Directory structure: default

Let's have a look how the directory is actually built up:

```
# Project directory
host> ls -l data/www/my-first-project/
total 4
drwxr-xr-x 2 cytopia cytopia 4096 Mar 12 23:05 htdocs/

# htdocs directory inside your project directory
host> ls -l data/www/my-first-project/htdocs
total 4
-rw-r--r-- 1 cytopia cytopia 87 Mar 12 23:05 index.php
```

By calling your project url, the index.php file will be served.

Directory structure: nested symlink

Most of the time you would clone or otherwise download a PHP framework, which in most cases has its own www directory somewhere nested. How can this be linked to the htdocs directory?

Let's have a look how the directory is actually built up:

```
# Project directory
host> ls -l data/www/my-first-project/
total 4
drwxr-xr-x 2 cytopia cytopia 4096 Mar 12 23:05 cakephp/
lrwxrwxrwx 1 cytopia cytopia 15 Mar 17 09:36 htdocs -> cakephp/webroot/

# htdocs directory inside your project directory
host> ls -l data/www/my-first-project/htdocs
total 4
-rw-r--r-- 1 cytopia cytopia 87 Mar 12 23:05 index.php
```

As you can see, the web server is still able to server the files from the `htdocs` location, this time however, `htdocs` itself is a symlink pointing to a much deeper and nested location inside an actual framework directory.

HTTPD_TEMPLATE_DIR

This variable specifies the directory name (which is just in your project directory, next to the `HTTPD_DOCROOT_DIR` directory) in which you can hold custom web server configuration files.

Every virtual host (which represents a project) can be fully customized to its own needs, independently of other virtual hosts.

This directory does not exist by default and you need to create it. Additionally you will also have to populate it with one of three yaml-based template files.

| Name | Allowed values | Default value |
|---------------------------------|----------------|------------------------|
| <code>HTTPD_TEMPLATE_DIR</code> | valid dir name | <code>.devilbox</code> |

Let's have a look at an imaginary project directory called `my-first-project`:

```
# Project directory
host> ls -l data/www/my-first-project/
total 4
drwxr-xr-x 2 cytopia cytopia 4096 Mar 12 23:05 htdocs/
```

Inside this your project directory you will need to create another directory which is called `.devilbox` by default. If you change the `HTTPD_TEMPLATE_DIR` variable to something else, you will have to create a directory by whatever name you chose for that variable.

```
# Project directory
host> cd data/www/my-first-project/
host> mkdir .devilbox
host> ls -l
total 4
drwxr-xr-x 2 cytopia cytopia 4096 Mar 12 23:05 .devilbox/
drwxr-xr-x 2 cytopia cytopia 4096 Mar 12 23:05 htdocs/
```

Now you need to copy the `vhost-gen` templates into the `.devilbox` directory. The templates are available in the Devilbox git directory under `templates/vhost-gen/`.

By copying those files into your project template directory, nothing will change, these are the default templates that will create the virtual host exactly the same way as if they were not present.

```
# Navigate into the devilbox directory
host> cd path/to/devilbox

# Copy templates to your project directory
host> cp templates/vhost-gen/* data/www/my-first-project/.devilbox/
```

Let's have a look how the directory is actually built up:

```
# Project directory
host> ls -l data/www/my-first-project/
total 4
drwxr-xr-x 2 cytopia cytopia 4096 Mar 12 23:05 .devilbox/
drwxr-xr-x 2 cytopia cytopia 4096 Mar 12 23:05 htdocs/
```

(continues on next page)

(continued from previous page)

```
# template directory inside your project directory
host> ls -l data/www/my-first-project/htdocs/.devilbox
total 4
-rw-r--r-- 1 cytopia cytopia 87 Mar 12 23:05 apache22.yml
-rw-r--r-- 1 cytopia cytopia 87 Mar 12 23:05 apache24.yml
-rw-r--r-- 1 cytopia cytopia 87 Mar 12 23:05 nginx.yml
```

The three files `apache22.yml`, `apache24.yml` and `nginx.yml` let you customize your web servers virtual host to anything from adding rewrite rules, overwriting directory index to even changing the server name or adding locations to other assets.

See also:

The whole process is based on a project called `vhost-gen`. A virtual host generator for Apache 2.2, Apache 2.4 and any Nginx version.

See also:

Customize your virtual host When you want to find out more how to actually customize each virtual host to its own need, read up more on *Customized virtual host (vhost-gen)*.

Tutorials Also have a look at this tutorial which is a walk-through showing you how to modify a virtual host and make it serve all files for multiple sub domains (server names): *Adding Sub domains*

46.6.3 MySQL

MYSQL_ROOT_PASSWORD

If you start a MySQL container for the first time, it will setup MySQL itself with this specified password. If you do change the root password to something else, make sure to also set it accordingly in `.env`, otherwise the devilbox will not be able to connect to MySQL and will not be able to display information inside the bundled intranet.

| Name | Allowed values | Default value |
|---------------------|----------------|---------------------|
| MYSQL_ROOT_PASSWORD | any string | empty (no password) |

Warning: Keep this variable in sync with the actual MySQL root password.

MYSQL_GENERAL_LOG

This variable controls the logging behaviour of the MySQL server (MySQL, MariaDB and PerconaDB). As the Devilbox is intended to be used for development, this feature is turned on by default.

| Name | Allowed values | Default value |
|-------------------|----------------|---------------|
| MYSQL_GENERAL_LOG | 0 or 1 | 0 |

MySQL documentation: “The general query log is a general record of what mysqld is doing. The server writes information to this log when clients connect or disconnect, and it logs each SQL statement received from clients. The general query log can be very useful when you suspect an error in a client and want to know exactly what the client sent to mysqld.”

– <https://dev.mysql.com/doc/refman/5.7/en/query-log.html>

46.6.4 PostgreSQL

PGSQL_ROOT_USER

If you start a PostgreSQL container for the first time, it will setup PostgreSQL itself with a specified username and password. If you do change the root username or password to something else, make sure to also set it accordingly in `env`, otherwise the devilbox will not be able to connect to PostgreSQL and will not be able to display information inside the bundled intranet.

| Name | Allowed values | Default value |
|-----------------|---------------------|---------------|
| PGSQL_ROOT_USER | alphabetical string | postgres |

Warning: Keep this variable in sync with the actual PostgreSQL username.

PGSQL_ROOT_PASSWORD

If you start a PostgreSQL container for the first time, it will setup PostgreSQL itself with a specified username and password. If you do change the root username or password to something else, make sure to also set it accordingly in `env`, otherwise the devilbox will not be able to connect to PostgreSQL and will not be able to display information inside the bundled intranet.

| Name | Allowed values | Default value |
|---------------------|----------------|---------------------|
| PGSQL_ROOT_PASSWORD | any string | empty (no password) |

Warning: Keep this variable in sync with the actual PostgreSQL password.

46.6.5 Bind

BIND_DNS_RESOLVER

This variable holds a comma separated list of IP addresses of DNS servers. By default using Google's DNS server as they are pretty fast.

| Name | Allowed values | Default value |
|-------------------|--------------------------------------|------------------|
| BIND_DNS_RESOLVER | comma separated list of IP addresses | 8.8.8.8, 8.8.4.4 |

The devilbox is using its own DNS server internally (your host computer can also use it for Auto-DNS) in order to resolve custom project domains defined by `TLD_SUFFIX`. To also be able to reach the internet from within the Container there must be some kind of upstream DNS server to ask for queries.

Some examples:

```
BIND_DNS_RESOLVER='8.8.8.8'
BIND_DNS_RESOLVER='8.8.8.8,192.168.0.10'
```

Note: If you don't trust the Google DNS server, then set it to something else. If you already have a DNS server inside your LAN and also want your custom DNS (if any) to be available inside the containers, set the value to its IP address.

BIND_DNSSEC_VALIDATE

This variable controls the DNSSEC validation of the DNS server. By default it is turned off.

| Name | Allowed values | Default value |
|----------------------|----------------|---------------|
| BIND_DNSSEC_VALIDATE | no, auto, yes | no |

- `yes` - DNSSEC validation is enabled, but a trust anchor must be manually configured. No validation will actually take place.
- `no` - DNSSEC validation is disabled, and recursive server will behave in the “old fashioned” way of performing insecure DNS lookups, until you have manually configured at least one trusted key.
- `auto` - DNSSEC validation is enabled, and a default trust anchor (included as part of BIND) for the DNS root zone is used.

BIND_LOG_DNS

This variable controls if DNS queries should be shown in Docker log output or not. By default no DNS queries are shown.

| Name | Allowed values | Default value |
|--------------|----------------|---------------|
| BIND_LOG_DNS | 1 or 0 | 0 |

If enabled all DNS queries are shown. This is useful for debugging.

BIND_TTL_TIME

This variable controls the DNS TTL in seconds. If empty or removed it will fallback to a sane default.

| Name | Allowed values | Default value |
|---------------|----------------|---------------|
| BIND_TTL_TIME | integer | empty |

See also:

- [BIND TTL](#)
- [BIND SOA](#)

BIND_REFRESH_TIME

This variable controls the DNS Refresh time in seconds. If empty or removed it will fallback to a sane default.

| Name | Allowed values | Default value |
|-------------------|----------------|---------------|
| BIND_REFRESH_TIME | integer | empty |

See also:

[BIND SOA](#)

BIND_RETRY_TIME

This variable controls the DNS Retry time in seconds. If empty or removed it will fallback to a sane default.

| Name | Allowed values | Default value |
|-----------------|----------------|---------------|
| BIND_RETRY_TIME | integer | empty |

See also:

[BIND SOA](#)

BIND_EXPIRY_TIME

This variable controls the DNS Expiry time in seconds. If empty or removed it will fallback to a sane default.

| Name | Allowed values | Default value |
|------------------|----------------|---------------|
| BIND_EXPIRY_TIME | integer | empty |

See also:

[BIND SOA](#)

BIND_MAX_CACHE_TIME

This variable controls the DNS Max Cache time in seconds. If empty or removed it will fallback to a sane default.

| Name | Allowed values | Default value |
|---------------------|----------------|---------------|
| BIND_MAX_CACHE_TIME | integer | empty |

See also:

[BIND SOA](#)

docker-compose.yml

This file is the core of the Devilbox and glues together all Docker images.

It is very tempting to just change this file in order to add new services to the already existing once. However your git directory will become dirty and you will always have to stash your changes before pulling new features from remote. To overcome this Docker Compose offers a default override file (`docker-compose.override.yml`) that let's you specify custom changes as well as completely new services without having to touch the default `docker-compose.yml`.

See also:

To find out more read [*docker-compose.override.yml*](#)

docker-compose.override.yml

The `docker-compose.override.yml` is the configuration file where you can override existing settings from `docker-compose.yml` or even add completely new services.

By default, this file does not exist and you must create it. You can either copy the existing `docker-compose.override.yml-example` or create a new one.

Table of Contents

- *Create docker-compose.override.yml*
 - *Copy example file*
 - *Create new file from scratch*
- *Further reading*

See also:

Official Docker documentation: [Share Compose configurations between files and projects](#)

48.1 Create docker-compose.override.yml

48.1.1 Copy example file

```
host> cd path/to/devilbox
host> cp docker-compose.override.yml-example docker-compose.override.yml
```

48.1.2 Create new file from scratch

1. Create an empty file within the Devilbox git directory named `docker-compose.override.yml`

2. Retrieve the currently used version from the existing `docker-compose.yml` file
3. Copy this version line to your newly created `docker-compose.override.yml` at the very top

```
# Create an empty file
host> cd path/to/devilbox
host> touch docker-compose.override.yml

# Retrieve the current version
host> grep ^version docker-compose.yml
version: '2.1'

# Add this version line to docker-compose.override.yml
host> echo "version: '2.1'" > docker-compose.override.yml
```

Let's see again how this file should look like now:

Listing 1: `docker-compose.override.yml`

```
version: '2.1'
```

Note: The documentation might be outdated and the version number might already be higher. Rely on the output of the `grep` command.

48.2 Further reading

To dive deeper into this topic and see how to actually add new services or overwrite existing services follow the below listed links:

See also:

- [*Add your own Docker image*](#)
- [*Overwrite existing Docker image*](#)

CHAPTER 49

apache.conf

Apache 2.2 and Apache 2.4 both come with their default vendor configuration. This might not be the ideal setup for some people, so you have the chance to change any of those settings, by supplying custom configurations.

See also:

If you are rather using Nginx, have a look at: [nginx.conf](#)

Important: You could actually also create virtual hosts here, but it is recommended to use the Devilbox Auto-vhost generation feature. If you want to customize your current virtual hosts have a look at [Customized virtual host \(vhost-gen\)](#).

Table of Contents

- [General](#)
- [Examples](#)
 - [Adjust KeepAlive settings for Apache 2.2](#)
 - [Limit HTTP headers and GET size for Apache 2.4](#)

49.1 General

You can set custom apache.conf configuration options for each Apache version separately. See the directory structure for Apache configuration directories inside `./cfg/` directory:

```
host> ls -l path/to/devilbox/cfg/ | grep 'apache'

drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 apache-2.2/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 apache-2.4/
```

Customization is achieved by placing a file into `cfg/apache-X.X/` (where `X.X` stands for your Apache version). The file must end by `.conf` in order to be sourced by the web server.

Each of the Apache configuration directories already contain an example file: `devilbox-custom.conf-example`, that can simply be renamed to `devilbox-custom.conf`. This file holds some example values that can be adjusted or commented out.

In order for the changes to be applied, you will have to restart the Devilbox.

49.2 Examples

49.2.1 Adjust KeepAlive settings for Apache 2.2

The following examples shows you how to change the `KeepAlive`, the `MaxKeepAliveRequests` as well as the `KeepAliveTimeout` values of Apache 2.2.

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Navigate to Apache 2.2 config directory
host> cd cfg/apache-2.2

# Create new conf file
host> touch keep_alive.conf
```

Now add the following content to the file:

Listing 1: `keep_alive.conf`

```
KeepAlive On
KeepAliveTimeout 10
MaxKeepAliveRequests 100
```

In order to apply the changes you need to restart the Devilbox.

Note: The above is just an example demonstration, you probably need other values for your setup. So make sure to understand how to configure Apache, if you are going to change any of those settings.

49.2.2 Limit HTTP headers and GET size for Apache 2.4

The following examples shows you how to limit the amount of headers the client can send to the server as well as changing the maximum URL GET size by adjusting `LimitRequestFields`, `LimitRequestFieldSize` and `LimitRequestLine` for Apache 2.4.

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Navigate to Apache 2.4 config directory
host> cd cfg/apache-2.4

# Create new conf file
host> touch limits.conf
```

Now add the following content to the file:

Listing 2: limits.conf

```
# Limit amount of HTTP headers a client can send to the server
LimitRequestFields 20
LimitRequestFieldSize 4094

# URL GET size
LimitRequestLine 2048
```

In order to apply the changes you need to restart the Devilbox.

Note: The above is just an example demonstration, you probably need other values for your setup. So make sure to understand how to configure Apache, if you are going to change any of those settings.

Nginx stable and Nginx mainline both come with their default vendor configuration. This might not be the ideal setup for some people, so you have the chance to change any of those settings, by supplying custom configurations.

See also:

If you are rather using Apache, have a look at: [apache.conf](#)

Important: You could actually also create virtual hosts here, but it is recommended to use the Devilbox Auto-vhost generation feature. If you want to customize your current virtual hosts have a look at [Customized virtual host \(vhost-gen\)](#).

Table of Contents

- [General](#)
- [Examples](#)
 - [Adjust KeepAlive settings for Nginx stable](#)
 - [Adjust timeout settings for Nginx mainline](#)

50.1 General

You can set custom nginx.conf configuration options for each Nginx version separately. See the directory structure for Nginx configuration directories inside `./cfg/` directory:

```
host> ls -l path/to/devilbox/cfg/ | grep 'nginx'

drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 nginx-mainline/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 nginx-stable/
```

Customization is achieved by placing a file into `cfg/nginx-X/` (where X stands for your Nginx flavour). The file must end by `.conf` in order to be sourced by the web server.

Each of the Nginx configuration directories already contain an example file: `devilbox-custom.conf-example`, that can simply be renamed to `devilbox-custom.conf`. This file holds some example values that can be adjusted or commented out.

In order for the changes to be applied, you will have to restart the Devilbox.

50.2 Examples

50.2.1 Adjust KeepAlive settings for Nginx stable

The following examples shows you how to change the `keepalive`, the `keepalive_requests` as well as the `keepalive_timeout` values of Nginx stable.

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Navigate to Nginx stable config directory
host> cd cfg/nginx-stable

# Create new conf file
host> touch keep_alive.conf
```

Now add the following content to the file:

Listing 1: `keep_alive.conf`

```
keepalive 10;
keepalive_timeout 10s;
keepalive_requests 100;
```

In order to apply the changes you need to restart the Devilbox.

Note: The above is just an example demonstration, you probably need other values for your setup. So make sure to understand how to configure Nginx, if you are going to change any of those settings.

50.2.2 Adjust timeout settings for Nginx mainline

The following examples shows you how to adjust various timeout settings for Nginx mainline by adjusting `client_body_timeout`, `client_header_timeout` and `send_timeout` directives.

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Navigate to Nginx mainline config directory
host> cd cfg/nginx-mainline

# Create new conf file
host> touch timeouts.conf
```

Now add the following content to the file:

Listing 2: timeouts.conf

```
client_body_timeout 60s;  
client_header_timeout 60s;  
send_timeout 60s;
```

In order to apply the changes you need to restart the Devilbox.

Note: The above is just an example demonstration, you probably need other values for your setup. So make sure to understand how to configure Nginx, if you are going to change any of those settings.

php.ini changes are global to all projects, but will only affect the currently selected PHP version.

Table of Contents

- *General*
- *Examples*
 - *Change memory_limit for PHP 7.1*
 - *Change timeout values for PHP 5.6*

51.1 General

You can set custom php.ini configuration options for each PHP version separately. See the directory structure for PHP configuration directories inside `./cfg/` directory:

```
host> ls -l path/to/devilbox/cfg/ | grep 'php-ini'

drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-ini-5.4/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-ini-5.5/
drwxr-xr-x  2 cytopia cytopia 4096 Apr  3 22:04 php-ini-5.6/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-ini-7.0/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-ini-7.1/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-ini-7.2/
```

Customization is achieved by placing a file into `cfg/php-ini-X.X/` (where X.X stands for your PHP version). The file must end by `.ini` in order to be sourced by the PHP-FPM server.

Each of the PHP ini configuration directories already contain an example file: `devilbox-custom.ini-example`, that can simply be renamed to `devilbox-custom.ini`. This file holds some example values that can be adjusted or commented out.

In order for the changes to be applied, you will have to restart the Devilbox.

51.2 Examples

51.2.1 Change memory_limit for PHP 7.1

The following examples shows you how to change the `memory_limit` of PHP 7.1 to 4096 MB.

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Navigate to PHP 7.1 config directory
host> cd cfg/php-ini-7.1

# Create new ini file
host> touch memory_limit.ini
```

Now add the following content to the file:

Listing 1: memory_limit.ini

```
[PHP]
memory_limit = 4096M
```

In order to apply the changes you need to restart the Devilbox. You can validate that the changes have taken place by visiting the Devilbox intranet `phpinfo` page.

51.2.2 Change timeout values for PHP 5.6

The following examples shows you how to change the `max_execution_time` and `max_input_time` of PHP 5.6.

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Navigate to PHP 5.6 config directory
host> cd cfg/php-ini-5.6

# Create new ini file
host> touch timeouts.ini
```

Now add the following content to the file:

Listing 2: timeouts.ini

```
[PHP]
max_execution_time = 180
max_input_time     = 180
```

In order to apply the changes you need to restart the Devilbox. You can validate that the changes have taken place by visiting the Devilbox intranet `phpinfo` page.

php-fpm.conf changes are global to all projects, but will only affect the currently selected PHP version.

Table of Contents

- *General*
- *Examples*
 - *Change rlimit core for master process for PHP 7.1*
 - *Change child process on pool www for PHP 5.6*
 - *Set non-overwritable php.ini values for PHP 7.0*

52.1 General

You can set custom php-fpm.conf configuration options for each PHP version separately. These changes affect the PHP-FPM process itself, global as well as pool specific configuration can be set.

Note: The default PHP-FPM pool is called `www` in case you want to make changes to it.

See the directory structure for PHP-FPM configuration directories inside `./cfg/` directory:

```
host> ls -l path/to/devilbox/cfg/ | grep 'php-fpm'

drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-fpm-5.4/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-fpm-5.5/
drwxr-xr-x  2 cytopia cytopia 4096 Apr  3 22:04 php-fpm-5.6/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-fpm-7.0/
```

(continues on next page)

(continued from previous page)

```
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-fpm-7.1/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-fpm-7.2/
```

Customization is achieved by placing a file into `cfg/php-fpm-X.X/` (where `X.X` stands for your PHP version). The file must end by `.conf` in order to be sourced by the PHP-FPM server.

Each of the PHP-FPM conf configuration directories already contain an example file: `devilbox-custom.conf-example`, that can simply be renamed to `devilbox-custom.conf`. This file holds some example values that can be adjusted or commented out.

In order for the changes to be applied, you will have to restart the Devilbox.

See also:

To find out about all available PHP-FPM directives, global or pool specific have a look at its documentation: <https://secure.php.net/manual/en/install.fpm.configuration.php>

52.2 Examples

52.2.1 Change rlimit core for master process for PHP 7.1

The following examples shows you how to change the `rlimit_core` of PHP-FPM 7.1 master process to 100.

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Navigate to PHP 7.1 config directory
host> cd cfg/php-fpm-7.1

# Create new conf file
host> touch rlimit.conf
```

Now add the following content to the file:

Listing 1: `rlimit.conf`

```
[global]
rlimit_core = 100
```

Important: Note the `[global]` section.

In order to apply the changes you need to restart the Devilbox.

52.2.2 Change child process on pool `www` for PHP 5.6

The following examples shows you how to change the `pm`, `pm.max_children`, `pm.start_servers`, `pm.min_spare_servers` and `pm.max_spare_servers` of PHP-FPM 5.6 on pool `www`.

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Navigate to PHP 5.6 config directory
```

(continues on next page)

(continued from previous page)

```
host> cd cfg/php-fpm-5.6

# Create new conf file
host> touch www_server.conf
```

Now add the following content to the file:

Listing 2: www_server.conf

```
[www]
; Pool config
pm = dynamic
pm.max_children = 10
pm.start_servers = 3
pm.min_spare_servers = 2
pm.max_spare_servers = 5
```

Important: Note the [www] section.

In order to apply the changes you need to restart the Devilbox.

52.2.3 Set non-overwritable php.ini values for PHP 7.0

You can also set `php.ini` values that cannot be overwritten by `php.ini` or the `ini_set()` function of PHP. This might be useful to make sure a specific value is enforced and will not be changed by some PHP frameworks on-the-fly.

This is achieved by `php_admin_flag` and `php_admin_value` that are parsed directly to PHP-FPM.

See also:

<https://secure.php.net/manual/en/install.fpm.configuration.php>

The following example will disable built-in PHP functions globally and non-overwriteable for PHP 7.0.

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Navigate to PHP 7.0 config directory
host> cd cfg/php-fpm-7.0

# Create new conf file
host> touch admin.conf
```

Now add the following content to the file:

Listing 3: admin.conf

```
[www]
php_admin_value[disable_functions] = link,symlink,popen,exec,system,shell_exec
```

Important: Note the [www] section.

Important: This kind of setting only has affects PHP files served through PHP-FPM, when you run php on the command line, this setting will be ignored.

Important: Be aware that none of your projects can use the above disabled functions anymore. They will simply not exist for PHP 7.0 after that configuration took affect.

In order to apply the changes you need to restart the Devilbox.

my.cnf

my.ini changes are global to all projects, but will only affect the currently selected MySQL version.

Important: When using *Docker Toolbox* on Windows, *.cnf files must have read-only file permissions, otherwise they are not sourced by the MySQL server.

Make sure to `chmod 0444 *.cnf` after adding your values.

Table of Contents

- *General*
- *Examples*
 - *Change key_buffer_size for MySQL 5.5*
 - *Change timeout and packet size for PerconaDB 5.7*

53.1 General

You can set custom MySQL options via your own defined my.cnf files for each version separately. See the directory structure for MySQL configuration directories inside ./cfg/ directory:

```
host> ls -l path/to/devilbox/cfg/ | grep -E 'mysql|mariadb|percona'
```

```
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 mariadb-10.0/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 mariadb-10.1/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 mariadb-10.2/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 mariadb-10.3/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 mysql-5.5/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 mysql-5.6/
```

(continues on next page)

(continued from previous page)

```
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 mysql-5.7/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 mysql-8.0/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 percona-5.5/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 percona-5.6/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 percona-5.7/
```

Customization is achieved by placing a file into `cfg/mysql-X.X/`, `cfg/mariadb-X.X/` or `cfg/percona-X.X` (where `X.X` stands for your MySQL version). The file must end by `.cnf` in order to be sourced by the MySQL server.

Each of the MySQL `cnf` configuration directories already contain an example file: `devilbox-custom.cnf-example`, that can simply be renamed to `devilbox-custom.cnf`. This file holds some example values that can be adjusted or commented out.

In order for the changes to be applied, you will have to restart the Devilbox.

53.2 Examples

53.2.1 Change `key_buffer_size` for MySQL 5.5

The following examples shows you how to change the `key_buffer_size` of MySQL 5.5 to 16 MB.

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Navigate to MySQL 5.5 config directory
host> cd cfg/mysql-5.5

# Create new cnf file
host> touch key_buffer_size.cnf
```

Now add the following content to the file:

Listing 1: `memory_limit.cnf`

```
[mysqld]
key_buffer_size=16M
```

In order to apply the changes you need to restart the Devilbox. You can validate that the changes have taken place by visiting the Devilbox intranet MySQL info page.

53.2.2 Change timeout and packet size for PerconaDB 5.7

The following examples shows you how to change the `wait_timeout` and `max_allowed_packet` of PerconaDB 5.7

```
# Navigate to the Devilbox directory
host> cd path/to/devilbox

# Navigate to PerconaDB 5.7 config directory
host> cd cfg/percona-5.7

# Create new ini file
host> touch timeouts.cnf
```

Now add the following content to the file:

Listing 2: timeouts.cnf

```
[mysqld]  
max_allowed_packet=256M  
wait_timeout = 86400
```

In order to apply the changes you need to restart the Devilbox. You can validate that the changes have taken place by visiting the Devilbox intranet MySQL info page.

Each PHP container is using bash as its default shell. If you do not like the way it is currently configured, you can add your own configuration files to overwrite settings.

See also:

Work inside the container

Table of Contents

- *Directory mapping*
- *Examples*
 - *Custom aliases*
 - *Custom vim configuration*

54.1 Directory mapping

Inside the Devilbox git directory you will find a directory called `bash/`. Every file inside this directory ending by `*.sh` will be source by your bash shell, allowing for a customized bash configuration. All files not ending by `*.sh` will be ignored and can be used to create config files for other programs.

The `bash/` directory will be mounted into the PHP container to `/etc/bashrc-devilbox.d/`.

| Host OS path | Docker path |
|----------------------|--------------------------------------|
| <code>./bash/</code> | <code>/etc/bashrc-devilbox.d/</code> |

54.2 Examples

54.2.1 Custom aliases

Let's say you want to add some custom shell aliases. All you have to do is create any file ending by `.sh` and place it into the `./bash/` directory:

```
# Navigate to the Devilbox git directory
host> cd path/to/devilbox

# Create a new file
host> touch ./bash/aliases.sh

# Add some content to the file
host> vi ./bash/aliases.sh
```

Listing 1: `./bash/aliases.sh`

```
alias l='ls -a'
alias ll='ls -al'
alias www='cd /shared/httpd'
```

54.2.2 Custom vim configuration

The `.vimrc` is usually place directly in the users home directory and the Devilbox does not offer any mounts directly to that directory, however you can use a trick with shell aliases to use `vim` with a different config file by default.

First of all, place your favorite `.vimrc` into the `./bash/` directory

```
# Navigate to the Devilbox git directory
host> cd path/to/devilbox

# Copy your vim config to the ./bash directory
host> cp ~/.vimrc bash/.vimrc
```

Right now, this is not going to do anything and as `.vimrc` is not ending by `.sh` it is also ignored by the shell itself. What is now left to do, is make `vim` itself always use this config file.

As you can see from the above stated directory mapping, the `.vimrc` file will end up under: `/etc/bashrc-devilbox.d/.vimrc` inside the PHP container, so just create a shell alias for `vim` that will always use this file:

```
# Navigate to the Devilbox git directory
host> cd path/to/devilbox

# Create a new file
host> touch ./bash/vim.sh

# Add your vim alias
host> vi ./bash/vim.sh
```


Listing 2: ./bash/vim.sh

```
alias vim='vim -u /etc/bashrc-devilbox.d/.vimrc'
```

Whenever you start `vim` inside any PHP container, it will automatically use the provided vim configuration file.

This trick will work for all tools that require configuration files.

Docker and Docker Compose

This section gives you some detail about installing Docker and Docker Compose on your operating system.

- *Install Docker*
 - *Docker on Linux*
 - *Docker on Windows*
 - *Docker on MacOS*
 - *docker user group*
- *Install Docker Compose*
- *Checklist*

55.1 Install Docker

See also:

Docker Toolbox Note, this section refers to **native Docker**, which is the recommended version. If however, you need to install **Docker Toolbox** (such as on Windows 7 or older Macs), have a look at this page.

Warning: The minimum required Docker version is 1.12.0. Make sure not to install older versions.

55.1.1 Docker on Linux

Refer to the official [Docker for Linux documentation](#) for how to install Docker on your specific Linux distribution.

55.1.2 Docker on Windows

Refer to the official [Docker for Windows documentation](#) for how to install Docker on Windows.

55.1.3 Docker on MacOS

Refer to the official [Docker for Mac documentation](#) for how to install Docker on MacOS.

55.1.4 docker user group

Docker itself requires super user privileges which is granted to a system wide group called `docker`. After having installed Docker on your system, ensure that your local user is assigned to the `docker` group. Check this via `groups` or `id` command.

```
host> id  
  
uid=1000(cytopia) gid=1000(cytopia) groups=1000(cytopia),999(docker)
```

55.2 Install Docker Compose

Warning: The minimum required Docker Compose version is 1.9.0. Make sure not to install older versions.

The Docker documentation provides various ways to install Docker Compose for all supported operating systems and is quite extensive and straight forward. Follow their steps here: [Install Docker Compose](#).

55.3 Checklist

1. Docker is installed at the minimum required version
2. Your user is part of the `docker` group
3. Docker Compose is installed at the minimum required version

- *Installation*
 - *Docker Toolbox on Windows*
 - *Docker Toolbox on MacOS*
 - *Docker Compose*
- *Additional steps*
 - *Listening address*
 - *Port forwarding*
 - *Auto-DNS*
- *Checklist*

56.1 Installation

Warning: The minimum required Docker Toolbox version is 1.12.0. Make sure not to install older versions.

56.1.1 Docker Toolbox on Windows

Refer to the official [Docker Toolbox on Windows documentation](#) for how to install Docker Toolbox on Windows.

56.1.2 Docker Toolbox on MacOS

Refer to the official [Docker Toolbox on MacOS documentation](#) for how to install Docker Toolbox on MacOS.

56.1.3 Docker Compose

When installing Docker Compose, make sure you do that **inside the virtual machine**.

See also:

Install Docker Compose Have a look at this page to help you install Docker Compose for your operating system.

56.2 Additional steps

Docker Toolbox is a legacy solution to bring Docker to systems which don't natively support Docker itself. This is achieved by starting a virtualized Linux (e.g.: on VirtualBox) and have Docker run inside.

You don't have to take care about setting up the virtual machine, this is done automatically with the provided setup file (Windows and MacOS).

This however has several disadvantages as the forwarded Docker ports are only visible inside the virtualized Linux and not on the host computer. Therefore the web server port cannot be reached on your host machine and you are not able to view any projects.

56.2.1 Listening address

First thing you need to make sure is that the `LOCAL_LISTEN_ADDR` variable from your `.env` file is empty. When it is empty all services bind to all IP addresses inside the virtual machine and thus being able to be seen from outside (your host operating system).

You can verify that the variable is actually empty by checking your `.env` file:

```
host> grep ^LOCAL_LISTEN_ADDR .env  
  
LOCAL_LISTEN_ADDR=
```

56.2.2 Port forwarding

Additionally I would suggest that you port-forward the virtual machines port 80 (which itself points to the docker container inside) to your host computers `127.0.0.1` address. This way you can reach the devilbox via `http://127.0.0.1` or `http://localhost`.

If you do not port-forward it to your host machines localhost, you will have to adjust all project DNS entries that are described in this documentation to go to `127.0.0.1` to the IP address of your virtual machine.

56.2.3 Auto-DNS

I am currently not aware that Auto-DNS will work with Docker Toolbox. If you are willing to spent some time there, let me know. There is currently an open ticket which is addressing this: <https://github.com/cytopia/devilbox/issues/101>

56.3 Checklist

1. Docker Toolbox is installed at minimum required version

2. Docker Compose is installed inside the virtual machine at minimum required version
3. LOCAL_LISTEN_ADDR is empty in the `.env` file

Available container

Note:

Start the Devilbox Find out how to start some or all container.

The following table gives you an overview about all container that can be started. When doing a selective start, use the Name value to specify the container to start up.

| Container | Name | Hostname | IP Address |
|---------------------------|-------|----------|----------------|
| DNS | bind | bind | 172.16.238.100 |
| PHP | php | php | 172.16.238.10 |
| Apache, Nginx | httpd | httpd | 172.16.238.11 |
| MySQL, MariaDB, PerconaDB | mysql | mysql | 172.16.238.12 |
| PostgreSQL | pgsql | pgsql | 172.16.238.13 |
| Redis | redis | redis | 172.16.238.14 |
| Memcached | memcd | memcd | 172.16.238.15 |
| MongoDB | mongo | mongo | 172.16.238.16 |

CHAPTER 58

Available tools

Each PHP container version brings the same tools, so you can safely switch versions without having to worry to have less or more tools available.

See also:

[Work inside the container](#)

The PHP container is your workhorse and these are your tools:

Note: If you are in need of other tools, open up an issue at [Github](#) and ask for it, this can usually be implemented very quickly.

See also:

If you ever feel those tools are out-dated, simply update your Docker images. Docker images are built every night to ensure latest tools and security patches: *[Update Docker images](#)*

Remove stopped container

59.1 Why should I?

If you simply `docker-compose stop` in order to stop all containers, they are still preserved in the `docker ps -a` process list and still have state.

In case you change any path variables inside the `.env` file (or silently due to git updates), you need to completely re-create the state.

This is done by first fully removing the container and then simply starting it again.

59.2 How to do it?

```
host> docker-compose stop
host> docker-compose rm
```

59.3 When to do it?

Whenever path values inside the `.env` file change.

Synchronize container permissions

One main problem with a running Docker container is to **synchronize the ownership of files in a mounted volume** in order to preserve security (Not having to use `chmod 0777` or root user).

This problem will be addressed below by using a PHP-FPM docker image as an example.

60.1 Unsynchronized permissions

Consider the following directory structure of a mounted volume. Your hosts computer uid/gid are 1000 which does not have a corresponding user/group within the container. Fortunately the `tmp/` directory allows everybody to create new files in it, because its permissions are `0777`.

| [Host] | | [Container] |
|--|--|---|
| ----- | | |
| <code>\$ ls -l</code> | | <code>\$ ls -l</code> |
| <code>-rw-r--r-- user group index.php</code> | | <code>-rw-r--r-- 1000 1000 index.php</code> |
| <code>drwxrwxrwx user group tmp/</code> | | <code>drwxrwxrwx 1000 1000 tmp/</code> |

Your web application might now have created some temporary files (via the PHP-FPM process) inside the `tmp/` directory:

| [Host] | | [Container] |
|--|--|--|
| ----- | | |
| <code>\$ ls -l tmp/</code> | | <code>\$ ls -l tmp/</code> |
| <code>-rw-r--r-- 96 96 _tmp_cache01.php</code> | | <code>-rw-r--r-- www www _tmp_cache01.php</code> |
| <code>-rw-r--r-- 96 96 _tmp_cache02.php</code> | | <code>-rw-r--r-- www www _tmp_cache01.php</code> |

On the Docker container side everything is still fine, but on your host computers side, those files now show a user id and group id of 96, which is in fact the uid/gid of the PHP-FPM process running inside the container. On the host side you have just lost write/delete access to those files and will now have to use `sudo` in order to delete/edit those files.

60.2 It gets even worse

Consider you had created the `tmp/` directory on your host only with `0775` permissions:

| [Host] | | [Container] |
|--|--|---|
| ----- | | |
| <code>\$ ls -l</code> | | <code>\$ ls -l</code> |
| <code>-rw-r--r-- user group index.php</code> | | <code>-rw-r--r-- 1000 1000 index.php</code> |
| <code>drwxrwxr-x user group tmp/</code> | | <code>drwxrwxr-x 1000 1000 tmp/</code> |

If your web application now wants to create some temporary files (via the PHP-FPM process) inside the `tmp/` directory, it will fail due to lack of permissions.

60.3 The solution

To overcome this problem, it must be made sure that the PHP-FPM process inside the container runs under the same uid/gid as your local user that mounts the volumes and also wants to work on those files locally. However, you never know during Image build time what user id this would be. Therefore it must be something that can be changed during startup of the container.

This is achieved in the Devilbox's containers by two environment variables that can be provided during startup in order to change the uid/gid of the PHP-FPM user prior starting up PHP-FPM process.

```
$ docker run -e NEW_UID=1000 -e NEW_GID=1000 -it devilbox/php-fpm:7.2-work
[INFO] Changing user 'devilbox' uid to: 1000
root $ usermod -u 1000 devilbox
[INFO] Changing group 'devilbox' gid to: 1000
root $ groupmod -g 1000 devilbox
[INFO] Starting PHP 7.2.0 (fpm-fcgi) (built: Oct 30 2017 12:05:19)
```

When `NEW_UID` and `NEW_GID` are provided to the startup command, the container will do a `usermod` and `groupmod` prior starting up in order to assign new uid/gid to the PHP-FPM user. When the PHP-FPM process finally starts up it actually runs with your local system user and making sure permissions will be in sync from now on.

Note: To tackle this on the PHP-FPM side is only half a solution to the problem. The same applies to a web server Docker container when you offer **file uploads**. They will be uploaded and created by the web servers uid/gid. Therefore the web server itself must also provide the same kind of solution.

Find common questions and answers here.

See also:

Troubleshooting

Table of Contents

- *General*
 - *Are there any differences between native Docker and Docker Toolbox?*
 - *Why are mounted MySQL data directories separated by version?*
 - *Why are mounted PostgreSQL data directories separated by version?*
 - *Why are mounted MongoDB data directories separated by version?*
 - *Why do the user/group permissions of log/ or cfg/ directories show 1000?*
 - *Can I not just comment out the service in the .env file?*
 - *Are there any required services that must/will always be started?*
 - *What PHP Modules are available?*
- *Configuration*
 - *Can I load custom PHP modules without rebuilding the Docker image?*
 - *Can I load custom Apache modules without rebuilding the Docker image?*
 - *Can I change the MySQL root password?*
 - *Can I change php.ini?*
 - *Can I change my.cnf?*
 - *Can I change the project virtual host domain .loc?*

- *Can I just start PHP and MySQL instead of all container?*
 - *Do I always have to edit `/etc/hosts` for new projects?*
- *Compatibility*
 - *Does it work with CakePHP?*
 - *Does it work with Drupal?*
 - *Does it work with Joomla?*
 - *Does it work with Laravel?*
 - *Does it work with Phalcon?*
 - *Does it work with Symfony?*
 - *Does it work with Wordpress?*
 - *Does it work with Yii?*
 - *Does it work with Zend?*

61.1 General

61.1.1 Are there any differences between native Docker and Docker Toolbox?

Yes, read *Docker Toolbox* to find out more.

61.1.2 Why are mounted MySQL data directories separated by version?

This is just a pre-caution. Imagine they would link to the same datadir. You start the Devilbox with mysql 5.5, create a database and add some data. Now you decide to switch to mysql 5.7 and restart the devilbox. The newer mysql version will probably upgrade the data leaving it unable to start with older mysql versions.

61.1.3 Why are mounted PostgreSQL data directories separated by version?

See: *Why are mounted MySQL data directories separated by version?*

61.1.4 Why are mounted MongoDB data directories separated by version?

See: *Why are mounted MySQL data directories separated by version?*

61.1.5 Why do the user/group permissions of log/ or cfg/ directories show 1000?

Uid and Gid are set to 1000 by default. You can alter them to match the uid/gid of your current user. Open the `.env` file and change the sections `NEW_UID` and `NEW_GID`. When you start up the devilbox, the PHP container will use these values for its user.

See also:

NEW_UID and *NEW_GID*

61.1.6 Can I not just comment out the service in the .env file?

No, don't do this. This will lead to unexpected behaviour (different versions will be loaded). The `.env` file allows you to configure the devilbox, but not to start services selectively.

61.1.7 Are there any required services that must/will always be started?

Yes. `http` and `php` will automatically always be started (due to dependencies inside `docker-compose.yml`) if you specify them or not.

61.1.8 What PHP Modules are available?

The Devilbox is a development stack, so it is made sure that a lot of PHP modules are available out of the box in order to work with many different frameworks.

Available PHP modules can be seen at the PHP Docker image repository.

See also:

<https://github.com/devilbox/docker-php-fpm>

61.2 Configuration

61.2.1 Can I load custom PHP modules without rebuilding the Docker image?

Yes, see `custom_php_modules`

61.2.2 Can I load custom Apache modules without rebuilding the Docker image?

Yes, see `custom_apache_modules`

61.2.3 Can I change the MySQL root password?

Yes, you can change the password of the MySQL root user. If you do so, you must also set the new password in your `.env` file. See `MYSQL_ROOT_PASSWORD` for how to change this value.

61.2.4 Can I change php.ini?

Yes, `php.ini` directives can be changed for each PHP version separately. See `php.ini`

61.2.5 Can I change my.cnf?

Yes, `my.cnf` directives can be changed for each MySQL version separately. See `my.cnf`

61.2.6 Can I change the project virtual host domain `.loc`?

Yes, the `.env` variable `TLD_SUFFIX` can be changed to whatever domain or subdomain you want. See [TLD_SUFFIX](#).

Warning: Be aware not to use `dev` or `localhost`. See [TLD_SUFFIX](#) for more details.

61.2.7 Can I just start PHP and MySQL instead of all container?

Yes, every Docker container is optional. The Devilbox allows for selective startup. See [Start the Devilbox](#).

61.2.8 Do I always have to edit `/etc/hosts` for new projects?

You need a valid DNS entry for every project that points to the Httpd server. As those records don't exist by default, you will have to create them. However, the Devilbox has a bundled DNS server that can automate this for you. The only thing you have to do for that to work is to add this DNS server's IP address to your `/etc/resolv.conf`. See [Auto-DNS](#) for detailed instructions.

61.3 Compatibility

61.3.1 Does it work with CakePHP?

Yes, see [Setup CakePHP](#)

61.3.2 Does it work with Drupal?

Yes, see [Setup Drupal](#)

61.3.3 Does it work with Joomla?

Yes, see [Setup Joomla](#)

61.3.4 Does it work with Laravel?

Yes, see [Setup Laravel](#)

61.3.5 Does it work with Phalcon?

Yes, see [Setup Phalcon](#)

61.3.6 Does it work with Symfony?

Yes, see [Setup Symfony](#)

61.3.7 Does it work with Wordpress?

Yes, see *Setup Wordpress*

61.3.8 Does it work with Yii?

Yes, see *Setup Yii*

61.3.9 Does it work with Zend?

Yes, see *Setup Zend*

This section will contain common problems and how to resolve them. It will grow over time once there are more issues reported.

See also:

[FAQ](#)

Table of Contents

- *Invalid bind mount spec*
- *[Warning] World-writable config file '/etc/mysql/docker-default.d/my.cnf' is ignored*

62.1 Invalid bind mount spec

This error might occur after changing the path of MySQL, PostgreSQL, Mongo or any other data directory.

When you change any paths inside `.env` that affect Docker mountpoints, the container needs to be removed and re-created during the next startup. Removing the container is sufficient as they will always be created during run if they don't exist.

In order to remove the container do the following:

```
host> cd path/to/devilbox
host> docker-compose stop

# Remove the stopped container (IMPORTANT!)
# After the removal it will be re-created during next run
host> docker-compose rm -f
```

See also:

[Remove stopped container](#)

62.2 [Warning] World-writable config file `/etc/mysql/docker-default.d/my.cnf` is ignored

This warning might occur when using *Docker Toolbox* on Windows and trying to apply custom MySQL configuration files. This will also result in the configuration file not being source by the MySQL server.

To fix this issue, you will have to change the file permission of your custom configuration files to read-only by applying the following `chmod` command.

```
# Nagivate to devilbox git directory
host> cd path/to/devilbox

# Navigate to the MySQL config directory (e.g.: MySQL 5.5)
host> cd cfg/mysql-5.5

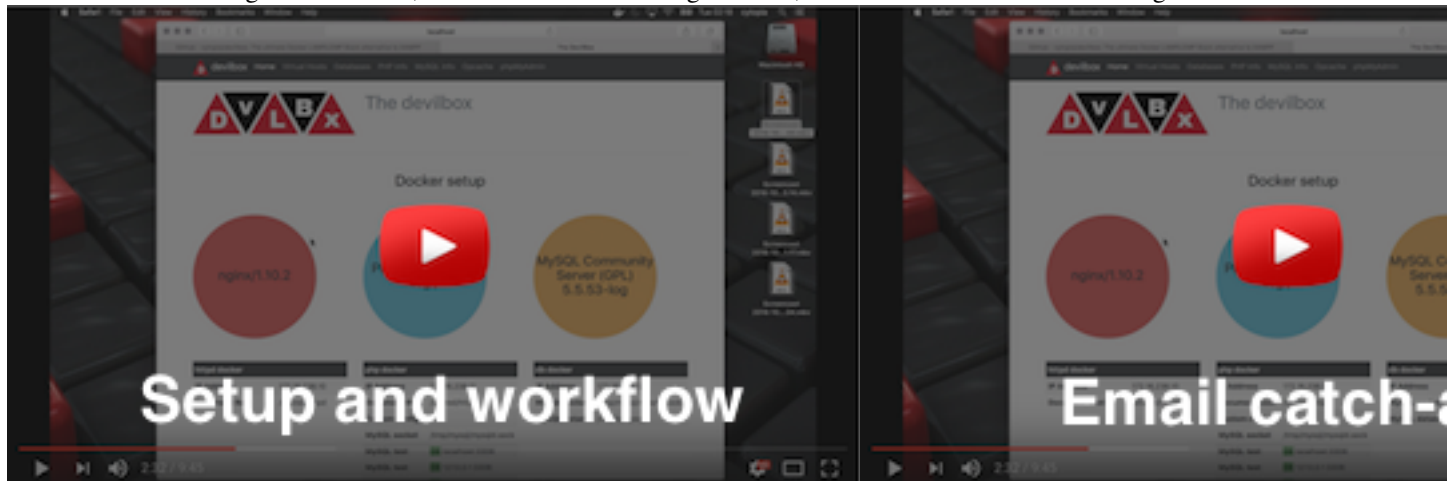
# Make cnf files read only
host> chmod 0444 *.cnf
```

See also:

- *my.cnf*
- <https://github.com/cytopia/devilbox/issues/212>

63.1 Official videos

The official videos might be a bit old, but are still valid and a good start, even if the intranet UI has changed a bit.



63.2 Blog posts

The following shows a list of blogs that give a nice and objective introduction to the Devilbox.

| Title | Language |
|---|----------|
| Using Devilbox For Local WordPress Development In Docker | English |
| Devilbox: Lokaler Webserver mit Apache, PHP und MySQL im Docker Container | German |

63.3 Use-cases

63.3.1 Joomla's Continuous Integration

Joomla has created a [PR Testing Platform](#) as their [Google Summer of Code 2017](#) project using a modified version of the Devilbox.

63.4 Add your story

Have you written a valuable blog about the Devilbox or do you have a fancy use-case? If so, submit a pull request and add it.

The Devilbox provides [official logos and banners](#) to be used for articles, blogs and others by the following license:



Images are available as opaque and transparent versions:



If you feel like designing a new logo for the Devilbox or just want to grab a copy of any of the images go to its artwork repository on github.

See also:

<https://github.com/devilbox/artwork>