

---

# Devilbox Documentation

*Release 1.0*

**cytopia**

**Jan 25, 2019**



---

## Contents

---

<b>1</b>	<b>Read first</b>	<b>3</b>
1.1	Shell commands . . . . .	3
1.2	Examples . . . . .	3
1.3	Checklists . . . . .	3
1.4	Where to start? . . . . .	4
<b>2</b>	<b>Features</b>	<b>5</b>
2.1	Projects . . . . .	6
2.2	Service and version choice . . . . .	7
2.3	Configuration . . . . .	7
2.4	Intranet . . . . .	8
2.5	Dockerized . . . . .	8
2.6	Others . . . . .	8
<b>3</b>	<b>Devilbox purpose</b>	<b>11</b>
3.1	Why did I built this? . . . . .	11
3.2	Automation is key . . . . .	11
3.3	Issues with Docker encountered . . . . .	12
3.4	Today's state . . . . .	12
3.5	Tomorrow's state . . . . .	12
<b>4</b>	<b>Prerequisites</b>	<b>13</b>
4.1	Supported host OS . . . . .	14
4.2	Required software . . . . .	14
4.3	Docker installation . . . . .	15
4.4	Post installation . . . . .	17
4.5	Optional previous knowledge . . . . .	17
<b>5</b>	<b>Install the Devilbox</b>	<b>19</b>
5.1	Download the Devilbox . . . . .	19
5.2	Create <code>.env</code> file . . . . .	20
5.3	Set uid and gid . . . . .	20
5.4	OS specific setup . . . . .	21
5.5	Checklist . . . . .	22
<b>6</b>	<b>Start the Devilbox</b>	<b>23</b>
6.1	The Devilbox startup explained . . . . .	23

6.2	Start all container . . . . .	24
6.3	Start some container . . . . .	24
6.4	Open Devilbox intranet . . . . .	25
6.5	Checklist . . . . .	26
<b>7</b>	<b>Devilbox intranet</b>	<b>27</b>
7.1	Devilbox tools . . . . .	28
7.2	Third-party tools . . . . .	28
7.3	Settings . . . . .	30
7.4	Checklist . . . . .	32
<b>8</b>	<b>Directory overview</b>	<b>35</b>
8.1	Data directory . . . . .	35
8.2	Project directory . . . . .	36
8.3	Docroot directory . . . . .	36
8.4	Domain suffix . . . . .	36
8.5	Making sense of it . . . . .	36
8.6	Checklist . . . . .	37
<b>9</b>	<b>Create your first project</b>	<b>39</b>
9.1	Step 1: visit Intranet vhost page . . . . .	39
9.2	Step 2: create a project directory . . . . .	40
9.3	Step 3: create a docroot directory . . . . .	40
9.4	Step 4: create a DNS entry . . . . .	42
9.5	Step 5: visit your project . . . . .	42
9.6	Step 6: create a hello world file . . . . .	43
9.7	Checklist . . . . .	44
9.8	Further examples . . . . .	44
<b>10</b>	<b>Enter the PHP container</b>	<b>45</b>
10.1	How to enter . . . . .	45
10.2	How to become root . . . . .	46
10.3	Tools . . . . .	46
10.4	Advanced . . . . .	47
10.5	Checklist . . . . .	47
<b>11</b>	<b>Change container versions</b>	<b>49</b>
11.1	Implications . . . . .	50
11.2	Examples . . . . .	50
11.3	Gotchas . . . . .	53
11.4	Checklist . . . . .	53
<b>12</b>	<b>Setup Auto DNS</b>	<b>55</b>
12.1	Native Docker . . . . .	55
12.2	Docker Toolbox . . . . .	57
<b>13</b>	<b>Setup valid HTTPS</b>	<b>59</b>
13.1	TL;DR . . . . .	59
13.2	How does it work . . . . .	60
13.3	Import the CA into your browser . . . . .	60
13.4	Further Reading . . . . .	61
<b>14</b>	<b>Configure PHP Xdebug</b>	<b>69</b>
14.1	Introduction . . . . .	69
14.2	Configure PHP container for Xdebug . . . . .	70

14.3	Configure your IDE/editor for Xdebug . . . . .	80
<b>15</b>	<b>Enable/disable PHP modules</b>	<b>89</b>
15.1	Enabled PHP modules . . . . .	89
15.2	Disable PHP modules . . . . .	90
15.3	Roadmap . . . . .	90
<b>16</b>	<b>Read log files</b>	<b>91</b>
16.1	Mounted logs . . . . .	91
16.2	Docker logs . . . . .	92
16.3	Checklist . . . . .	92
<b>17</b>	<b>Email catch-all</b>	<b>93</b>
17.1	Devilbox Intranet . . . . .	93
17.2	MailHog . . . . .	93
<b>18</b>	<b>Add custom environment variables</b>	<b>95</b>
18.1	Add custom environment variables . . . . .	95
18.2	Use custom environment variables . . . . .	96
<b>19</b>	<b>Work inside the PHP container</b>	<b>97</b>
19.1	Enter the container . . . . .	98
19.2	Inside the container . . . . .	98
19.3	Leave the container . . . . .	99
19.4	Host to Container mappings . . . . .	99
19.5	Checklist . . . . .	101
<b>20</b>	<b>Source Code Analysis</b>	<b>103</b>
20.1	Awesome-ci . . . . .	103
20.2	PHPCS . . . . .	104
20.3	ESLint . . . . .	104
<b>21</b>	<b>Best practice</b>	<b>107</b>
21.1	Move data out of Devilbox directory . . . . .	107
21.2	PHP project hostname settings . . . . .	110
21.3	Timezone . . . . .	110
<b>22</b>	<b>Customize PHP globally</b>	<b>113</b>
22.1	Configure PHP settings globally . . . . .	113
22.2	Configure non-overwritable settings globally . . . . .	114
22.3	Configure loaded PHP modules . . . . .	114
22.4	Configure PHP-FPM service . . . . .	114
<b>23</b>	<b>Customize web server globally</b>	<b>115</b>
23.1	Configure Apache . . . . .	115
23.2	Configure Nginx . . . . .	116
23.3	Devilbox specific settings . . . . .	116
<b>24</b>	<b>Connect to host OS</b>	<b>117</b>
24.1	Prerequisites . . . . .	117
24.2	Docker on Linux . . . . .	118
24.3	Docker for Mac . . . . .	118
24.4	Docker for Windows . . . . .	118
24.5	Docker Toolbox . . . . .	119

<b>25</b>	<b>Connect to other Docker container</b>	<b>121</b>
25.1	Any Docker container on host os . . . . .	121
25.2	Add Docker container to Devilbox network . . . . .	122
25.3	Add Docker container to Devilbox stack . . . . .	122
<b>26</b>	<b>Connect to external hosts</b>	<b>123</b>
<b>27</b>	<b>Add custom CNAME DNS entries</b>	<b>125</b>
27.1	Why and what? . . . . .	125
27.2	How? . . . . .	125
<b>28</b>	<b>Add your own Docker image</b>	<b>127</b>
28.1	Prerequisites . . . . .	127
28.2	What information do you need? . . . . .	128
28.3	How to add a new service? . . . . .	128
28.4	How to start the new service? . . . . .	130
28.5	Further reading . . . . .	130
<b>29</b>	<b>Overwrite existing Docker image</b>	<b>131</b>
29.1	Prerequisites . . . . .	131
29.2	What information do you need? . . . . .	131
29.3	How to overwrite a service? . . . . .	132
29.4	Further reading . . . . .	133
<b>30</b>	<b>Custom scripts per PHP version</b>	<b>135</b>
30.1	General . . . . .	135
30.2	Examples . . . . .	136
<b>31</b>	<b>Custom scripts globally</b>	<b>139</b>
31.1	General . . . . .	139
31.2	Examples . . . . .	140
<b>32</b>	<b>Autostarting NodeJS Apps</b>	<b>143</b>
32.1	Self-built . . . . .	143
32.2	Pre-built . . . . .	144
32.3	Reverse proxy NodeJS . . . . .	145
<b>33</b>	<b>Virtual host templates</b>	<b>147</b>
33.1	Overview . . . . .	148
33.2	Virtual host Templates . . . . .	149
33.3	Reverse proxy Templates . . . . .	158
<b>34</b>	<b>Customize all virtual hosts globally</b>	<b>167</b>
34.1	Prerequisite . . . . .	167
34.2	Apply templates globally to all vhosts . . . . .	167
<b>35</b>	<b>Customize specific virtual host</b>	<b>169</b>
35.1	vhost-gen . . . . .	170
35.2	Templates explained . . . . .	172
35.3	Apply Changes . . . . .	175
35.4	Further readings . . . . .	176
<b>36</b>	<b>Virtual host vs Reverse Proxy</b>	<b>177</b>
36.1	Motivation . . . . .	177
36.2	Benefits . . . . .	177
36.3	Creating a reverse proxy . . . . .	178

<b>37 Example: add sub domains</b>	<b>179</b>
37.1 Simple sub domains for one project . . . . .	180
37.2 Complex sub domains for one project . . . . .	181
<b>38 Reverse Proxy with HTTPS</b>	<b>191</b>
38.1 Walkthrough . . . . .	192
<b>39 Reverse Proxy for custom Docker</b>	<b>197</b>
39.1 Walkthrough . . . . .	198
<b>40 Enable all additional container</b>	<b>203</b>
40.1 Available additional container . . . . .	203
40.2 Enable all additional container . . . . .	203
40.3 Configure additional container . . . . .	204
<b>41 Enable and configure Blackfire</b>	<b>205</b>
41.1 Overview . . . . .	206
41.2 Instructions . . . . .	207
41.3 TL;DR . . . . .	208
<b>42 Enable and configure MailHog</b>	<b>209</b>
42.1 Overview . . . . .	210
42.2 Instructions . . . . .	211
42.3 TL;DR . . . . .	212
<b>43 Enable and configure RabbitMQ</b>	<b>213</b>
43.1 Overview . . . . .	214
43.2 Instructions . . . . .	215
43.3 TL;DR . . . . .	216
<b>44 Enable and configure Solr</b>	<b>217</b>
44.1 Overview . . . . .	218
44.2 Instructions . . . . .	219
44.3 TL;DR . . . . .	219
<b>45 Shared Devilbox server in LAN</b>	<b>221</b>
45.1 Prerequisites . . . . .	222
45.2 Project access . . . . .	222
45.3 Handle DNS records . . . . .	223
45.4 Share Devilbox CA . . . . .	225
<b>46 Use external databases</b>	<b>227</b>
46.1 Why . . . . .	227
46.2 Database on host os . . . . .	227
46.3 Database on network . . . . .	228
46.4 Database on internet . . . . .	228
<b>47 Checkout different Devilbox release</b>	<b>229</b>
<b>48 Remove stopped container</b>	<b>231</b>
48.1 Why should I? . . . . .	231
48.2 How to do it? . . . . .	231
48.3 When to do it? . . . . .	231
<b>49 Update the Devilbox</b>	<b>233</b>
49.1 Update git repository . . . . .	234

49.2	Update Docker images . . . . .	235
49.3	Checklist git repository . . . . .	236
49.4	Checklist Docker images . . . . .	236
<b>50</b>	<b>Remove the Devilbox</b>	<b>237</b>
50.1	Backups . . . . .	237
50.2	Remove the Devilbox . . . . .	238
50.3	Revert your system changes . . . . .	239
<b>51</b>	<b>Backup and restore MySQL</b>	<b>241</b>
51.1	Backup . . . . .	242
51.2	Restore . . . . .	245
<b>52</b>	<b>Backup and restore PostgreSQL</b>	<b>247</b>
52.1	Backup . . . . .	248
52.2	Restore . . . . .	248
<b>53</b>	<b>Backup and restore MongoDB</b>	<b>251</b>
53.1	Backup . . . . .	251
53.2	Restore . . . . .	252
<b>54</b>	<b>.env file</b>	<b>253</b>
54.1	Core settings . . . . .	255
54.2	Intranet settings . . . . .	261
54.3	Docker image versions . . . . .	262
54.4	Docker host mounts . . . . .	266
54.5	Docker host ports . . . . .	269
54.6	Container settings . . . . .	272
<b>55</b>	<b>docker-compose.yml</b>	<b>281</b>
<b>56</b>	<b>docker-compose.override.yml</b>	<b>283</b>
56.1	Create docker-compose.override.yml . . . . .	283
56.2	Further reading . . . . .	284
<b>57</b>	<b>apache.conf</b>	<b>285</b>
57.1	General . . . . .	285
57.2	Examples . . . . .	286
<b>58</b>	<b>nginx.conf</b>	<b>289</b>
58.1	General . . . . .	289
58.2	Examples . . . . .	290
<b>59</b>	<b>php.ini</b>	<b>293</b>
59.1	General . . . . .	293
59.2	Examples . . . . .	294
<b>60</b>	<b>php-fpm.conf</b>	<b>297</b>
60.1	General . . . . .	297
60.2	Examples . . . . .	298
<b>61</b>	<b>my.cnf</b>	<b>301</b>
61.1	General . . . . .	301
61.2	Examples . . . . .	302



<b>62</b>	<b>bashrc.sh</b>	<b>305</b>
62.1	Directory mapping . . . . .	305
62.2	Examples . . . . .	306
<b>63</b>	<b>Setup CakePHP</b>	<b>309</b>
63.1	Overview . . . . .	309
63.2	Walk through . . . . .	310
<b>64</b>	<b>Setup CodeIgniter</b>	<b>313</b>
64.1	Overview . . . . .	313
64.2	Walk through . . . . .	314
<b>65</b>	<b>Setup CraftCMS</b>	<b>317</b>
65.1	Overview . . . . .	317
65.2	Walk through . . . . .	318
<b>66</b>	<b>Setup Drupal</b>	<b>323</b>
66.1	Overview . . . . .	323
66.2	Walk through . . . . .	324
<b>67</b>	<b>Setup Joomla</b>	<b>327</b>
67.1	Overview . . . . .	327
67.2	Walk through . . . . .	328
<b>68</b>	<b>Setup Laravel</b>	<b>331</b>
68.1	Overview . . . . .	331
68.2	Walk through . . . . .	332
<b>69</b>	<b>Setup Magento 2</b>	<b>335</b>
69.1	Overview . . . . .	335
69.2	Walk through . . . . .	336
<b>70</b>	<b>Setup Phalcon</b>	<b>339</b>
70.1	Overview . . . . .	339
70.2	Walk through . . . . .	340
<b>71</b>	<b>Setup Photon CMS</b>	<b>343</b>
71.1	Overview . . . . .	343
71.2	Walk through . . . . .	344
<b>72</b>	<b>Setup PrestaShop</b>	<b>347</b>
72.1	Overview . . . . .	347
72.2	Walk through . . . . .	348
<b>73</b>	<b>Setup Shopware</b>	<b>351</b>
73.1	Overview . . . . .	351
73.2	Walk through . . . . .	352
73.3	Encountered problems . . . . .	354
<b>74</b>	<b>Setup Symfony</b>	<b>355</b>
74.1	Overview . . . . .	355
74.2	Walk through . . . . .	356
<b>75</b>	<b>Setup Typo3</b>	<b>359</b>
75.1	Overview . . . . .	359
75.2	Walk through . . . . .	360

<b>76 Setup Wordpress</b>	<b>363</b>
76.1 Overview . . . . .	363
76.2 Walk through . . . . .	364
<b>77 Setup Yii</b>	<b>373</b>
77.1 Overview . . . . .	373
77.2 Walk through . . . . .	374
<b>78 Setup Zend</b>	<b>377</b>
78.1 Overview . . . . .	377
78.2 Walk through . . . . .	378
<b>79 Setup other Frameworks</b>	<b>381</b>
<b>80 Setup reverse proxy NodeJS</b>	<b>383</b>
80.1 Overview . . . . .	384
80.2 Walk through . . . . .	384
80.3 Managing NodeJS . . . . .	391
<b>81 Setup reverse proxy Sphinx docs</b>	<b>393</b>
81.1 Overview . . . . .	394
81.2 Walk through . . . . .	394
<b>82 Synchronize container permissions</b>	<b>401</b>
82.1 Unsynchronized permissions . . . . .	401
82.2 It gets even worse . . . . .	402
82.3 The solution . . . . .	402
<b>83 Available container</b>	<b>403</b>
83.1 Core container . . . . .	403
83.2 Additional container . . . . .	403
<b>84 Available tools</b>	<b>405</b>
<b>85 Troubleshooting</b>	<b>407</b>
85.1 General . . . . .	408
85.2 Performance . . . . .	409
85.3 DNS issues . . . . .	410
85.4 SSL issues . . . . .	410
85.5 Web server issues . . . . .	410
85.6 PHP issues . . . . .	411
85.7 Database issues . . . . .	412
<b>86 FAQ</b>	<b>413</b>
86.1 General . . . . .	414
86.2 Configuration . . . . .	415
86.3 Compatibility . . . . .	416
<b>87 How To</b>	<b>419</b>
87.1 Add custom DNS server on Android . . . . .	419
87.2 Add custom DNS server on iPhone . . . . .	424
87.3 Add custom DNS server on Linux . . . . .	425
87.4 Add custom DNS server on MacOS . . . . .	430
87.5 Add custom DNS server on Windows . . . . .	431
87.6 Add project hosts entry on Linux . . . . .	431
87.7 Add project hosts entry on MacOS . . . . .	434

87.8	Add project hosts entry on Windows . . . . .	435
87.9	Find your user id and group id on MacOS . . . . .	437
87.10	Find your user id and group id on Windows . . . . .	438
87.11	Find Docker and Docker Compose version . . . . .	439
87.12	Move projects to a different directory . . . . .	439
87.13	Host address alias on MacOS . . . . .	440
87.14	Docker Toolbox and the Devilbox . . . . .	441
87.15	Find Docker Toolbox IP address . . . . .	444
87.16	SSH into Docker Toolbox . . . . .	445
87.17	SSH port-forward on Docker Toolbox from host . . . . .	447
87.18	SSH port-forward on host to Docker Toolbox . . . . .	449
87.19	Open a terminal on MacOS . . . . .	451
87.20	Open a terminal on Windows . . . . .	453
<b>88</b>	<b>Blogs, Videos and Use-cases</b>	<b>457</b>
88.1	Official videos . . . . .	457
88.2	Conferences . . . . .	457
88.3	Blog posts . . . . .	459
88.4	Use-cases . . . . .	459
88.5	Add your story . . . . .	459
<b>89</b>	<b>Artwork</b>	<b>461</b>





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 Examples

**Note:** Most examples to configure your host operating system will be presented for Linux by default, there will however always be links for how to accomplish the same on Windows and MacOS.

---

## 1.3 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.

---

## 1.4 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 SSL certificates*
  - *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 SSL certificates

Whenever a new project is created, SSL certificates are generated as well and assigned to that virtual host. Those certificates are signed by the Devilbox certificate authority which can be imported into your local browser to make all certificates valid and trusted.

### 2.1.4 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. This removes the need to create developer DNS records in `/etc/hosts`.

### 2.1.5 Email catch-all

All outgoing emails originating from your projects are intercepted, stored locally and can be viewed within the bundled intranet.

### 2.1.6 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.

### 2.1.7 Virtual host domains

Each of your virtual host will have its own domain. TLD can be freely chosen, such as `*.loc` or `*.local`. Be aware that some TLD's can cause problems. Read more here: [\*TLD\\_SUFFIX\*](#).

## 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 .

## 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: , , , and 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 basically just a `docker-compose.yml` file and you can easily add any Docker images you are currently missing in the Devilbox setup.



## CHAPTER 3

---

### Devilbox purpose

---

The Devilbox aims to provide you a universal zero-configuration LAMP and MEAN development environment for any purpose which is setup in less than 5 minutes.

Its main intention is to support an unlimited number of projects for any framework or cms and be portable accross all major operating systems, as well as providing any available php version with whatever module you require.

To be portable, customizable and as leight weight as possible, the choice fell on a Dockerized setup.

### 3.1 Why did I built this?

In one of my previous jobs I had to maintain around 30 different PHP projects. Many of them utilized different versions and configuration, thus I had to switch between my local MacOS and various Linux VMs on a frequent base in order to fullfill the current requirement.

Setting up new vhosts, local DNS entries, self-signed https certificates, installing other PHP versions, ensuring I had all modules and lots of other initial configuration was always a pain to me, so I decided to automate this.

### 3.2 Automation is key

A few month after releasing it on Github I hit another problem: Tickets regarding outdated versions as well as new major version requests accumulated and I spent a lot of time keeping up with updating and creating Docker images and making them available.

That was the point when I decided to create a fully automated and generalized build infrastructure for all custom Docker images.

The outcome was this:

- Docker images are generated and verified with Ansible
- Docker images have extensive CI tests
- Docker images are automatically built, tested and updated every night and pushed on success

### 3.3 Issues with Docker encountered

One of the major issues I have encountered with Docker is the synchronization of file and directory permissions between local and Docker mounted directories.

This is due to the fact that the process of PHP or the web server usually run with a different `uid` and `gid` as the local user starting the Docker container. Whenever a new file is created from inside the container, it will happen with the `uid` of the process running inside the container, thus making it incompatible with your local user.

This problem has been finally addressed with the Devilbox and you can read up on it in much more detail here: *[Synchronize container permissions](#)*.

### 3.4 Today's state

Honestly speaking, in the time I spent to build the Devilbox, I could have configured every possible VM by now, but I would have missed the fun. I learned a lot and in the end it made my work much more pleasant.

### 3.5 Tomorrow's state

I use the Devilbox on a daily base and together with other developers we find more and more edge cases that are being resolved. As technology also advanced quickly, the Devilbox needs to keep up with as well. Next major milestones will be to modularize it for easier customization of currently not available Container, hardening for production usage and workflows for deployments in a CI/CD landscape.



# CHAPTER 4

## Prerequisites

### Important:

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




### Table of Contents

- *Supported host OS*
- *Required software*
- *Docker installation*
  - *Linux*
  - *Mac*
    - \* *Docker for Mac*
    - \* *Docker Toolbox*
  - *Windows*
    - \* *Docker for Windows*
    - \* *Docker Toolbox*
- *Post installation*
  - *User settings*
  - *Shared drives*
  - *Network and firewall*
  - *SE Linux*
  - *General*

- *Optional previous knowledge*

## 4.1 Supported host OS

The Devilbox runs on all major operating systems which provide `Docker` and `Docker Compose`. See the matrix below for supported versions:

OS	Version	Type	Recommended
	Any		yes
	Any		yes
	Windows 7		yes
	Windows 10		yes
	Windows Server 2016		yes

## 4.2 Required software

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.

See also:

- 
- 
- 
- *Find Docker and Docker Compose version*

## 4.3 Docker installation

### 4.3.1 Linux



Docker on Linux 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 a member of the `docker` group.

```
host> id
uid=1000(cytopia) gid=1000(cytopia) groups=1000(cytopia),999(docker)
```

See also:

- 
- 
- 
- 
- (covers `docker` group)

### 4.3.2 Mac



On MacOS Docker is available in two different forms: **Docker for Mac** and **Docker Toolbox**.

#### Docker for Mac

Docker for Mac is the native and recommended version to choose when using the Devilbox.

Docker for Mac 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 a member of the `docker` group.

```
host> id
uid=502(cytopia) gid=20(staff) groups=20(staff),999(docker)
```

See also:

**Docker for Mac**

-

- 

## Docker Toolbox

If you still want to use Docker Toolbox, ensure you have read its drawbacks in the below provided links.

**See also:**

### Docker Toolbox

- 
- 
- 

---

**Important:** *Docker Toolbox and the Devilbox*

---

## 4.3.3 Windows



On Windows Docker is available in two different forms: **Docker for Windows** and **Docker Toolbox**.

### Docker for Windows

Docker for Windows is the native and recommended version to choose when using the Devilbox. This however is only available since **Windows 10**.

Docker for Windows requires administrative privileges which is granted to a system wide group called `docker-users`. After having installed Docker on your system, ensure that your local user is a member of the `docker-users` group.

**See also:**

### Docker for Windows

- 
- 

## Docker Toolbox

If you are on **Windows 7** or still want to use Docker Toolbox, ensure you have read its drawbacks in the below provided links.

**See also:**

### Docker Toolbox

-

- 

---

**Important:** *Docker Toolbox and the Devilbox*

---

## 4.4 Post installation

Read the Docker documentation carefully and follow all **install** and **post-install** steps. Below are a few stumbling blocks to check that might or might not apply depending on your host operating system and your Docker version.

**See also:**

*Troubleshooting*

### 4.4.1 User settings

Some versions of Docker require your local user to be in the `docker` group (or `docker-users` on Windows).

### 4.4.2 Shared drives

Some versions of Docker require you to correctly setup shared drives. Ensure the desired locations are being made available to Docker and the correct credentials are applied.

### 4.4.3 Network and firewall

On Windows, ensure your firewall allows access to shared drives.

### 4.4.4 SE Linux

Make sure to read any shortcomings when SE Linux is enabled.

### 4.4.5 General

It could also help to do a full system restart after the installation has been finished.

## 4.5 Optional previous knowledge

In order to easily work with the Devilbox you should already be familiar with the following:

- Navigate on the command line
- Docker Compose commands (, , , and )
- Docker Compose `.env` file
- Know how to use `git`

**See also:**

-

- 
- *Troubleshooting*

# CHAPTER 5

## Install the Devilbox

**Important:** Ensure you have read and followed the *Prerequisites*

### Table of Contents

- *Download the Devilbox*
- *Create .env file*
- *Set uid and gid*
  - *Find your user id*
  - *Find your group id*
- *OS specific setup*
  - *Linux: SELinux*
  - *OSX: Performance*
- *Checklist*

## 5.1 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
```

### See also:

- *Open a terminal on MacOS*

- *Open a terminal on Windows*
- *Checkout different Devilbox release*

## 5.2 Create .env file

Inside the cloned Devilbox git directory, you will find a file called `env-example`. This file is the main configuration with sane defaults for Docker Compose. In order to use it, it must be copied to a file named `.env`. (Pay attention to 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:

- 
- *.env file*

## 5.3 Set uid and gid

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:

### 5.3.1 Find your user id

```
host> id -u
```

### 5.3.2 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
```

See also:



- 
- *Find your user id and group id on MacOS*
- *Find your user id and group id on Windows*
- *Synchronize container permissions*

## 5.4 OS specific setup

### 5.4.1 Linux: SELinux

If you have SELinux enabled, you will also have to adjust the `MOUNT_OPTIONS` to allow shared mounts among multiple container:

Listing 2: .env

```
host> vi .env
MOUNT_OPTIONS=,z
```

**See also:**

- <https://github.com/cytopia/devilbox/issues/255>
- `MOUNT_OPTIONS`
- 
- 

### 5.4.2 OSX: Performance

Out of the box, Docker for Mac has some performance issues when it comes to mount directories with a lot of files inside. To mitigate this issue, you can adjust the caching settings for mounted directories.

To do so, you will want to adjust the `MOUNT_OPTIONS` to allow caching on mounts.

Listing 3: .env

```
host> vi .env
MOUNT_OPTIONS=,cached
```

Ensure to read the links below to understand why this problem exists and how the fix works. The Docker documentation will also give you alternative caching options to consider.

**See also:**

- <https://github.com/cytopia/devilbox/issues/105#issuecomment-426229921>
- <https://forums.docker.com/t/file-access-in-mounted-volumes-extremely-slow-cpu-bound/8076/281>
- <https://docs.docker.com/docker-for-mac/osxfuse-caching/#tuning-with-consistent-cached-and-delegated-configurations>
- `MOUNT_OPTIONS`

## 5.5 Checklist

1. Devilbox is cloned
2. `.env` file is created
3. 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.

**See also:**

*[Troubleshooting](#)*

## CHAPTER 6

---

### 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 `,` `,` `,` `,` and commands.

---

#### Table of Contents

- *The Devilbox startup explained*
- *Start all container*
  - *Foreground*
  - *Background*
- *Start some container*
  - *Foreground*
  - *Background*
- *Open Devilbox intranet*
- *Checklist*

### 6.1 The Devilbox startup explained

To gain a brief understanding about what is happening under the hood during startup, read ahead or skip directly to: *Start all container*.

Startup operations with the same configuration are idempotent, thus consecutive startups will not introduce any new changes. The following shows the brief startup steps:

- Docker Compose will automatically pull all necessary Docker images if they do not exist locally.
- Once the HTTPD container start, it will automatically create a Certificate Authority to be used for https connections and will place it in the `ca/` directory.
- The HTTPD container will then look for already available projects and create virtual hosts configurations, apply vhost-gen templates as well as CA-signed HTTPS certificates.
- Once the Bind container start, it will create a wildcard DNS zone for the given *TLD\_SUFFIX*
- In case MySQL or PostgreSQL container start, they will populate itself with their required default databases.

---

**Note:** Docker images are only pulled if they do not exist. They are not updated automatically. If you want to update to new Docker images read on: [Update the Devilbox](#).

---

## 6.2 Start all container

If you want all provided docker container to be available (as defined in `docker-compose.yml`), start them all by not explicitly specifying any image name.

### 6.2.1 Foreground

For the first startup, foreground start is recommended to see any errors that might occur:

```
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

### 6.2.2 Background

For consecutive startups you can send them into background (`-d`):

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

- If you want to gracefully stop all container, enter `docker-compose stop`
- If you want to kill all container, enter `docker-compose kill`

## 6.3 Start some container

If you don't require all container to be up and running and let's say just PHP, HTTPD and MySQL, you must explicitly specify the image names to start:

### 6.3.1 Foreground

```
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

## 6.3.2 Background

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

- If you want to gracefully stop all container, enter `docker-compose stop`
- If you want to kill all container, enter `docker-compose kill`

### 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.

## 6.4 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:

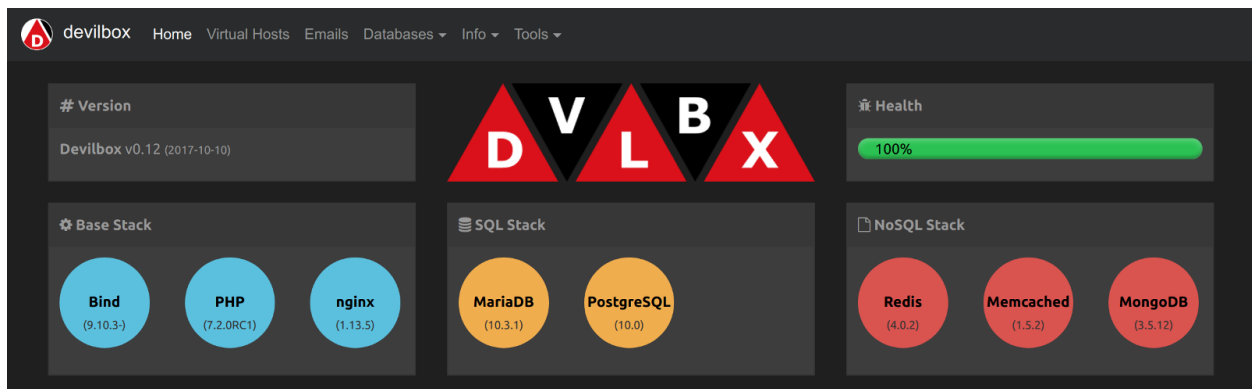


Fig. 1: Devilbox intranet: index dash view for all started container

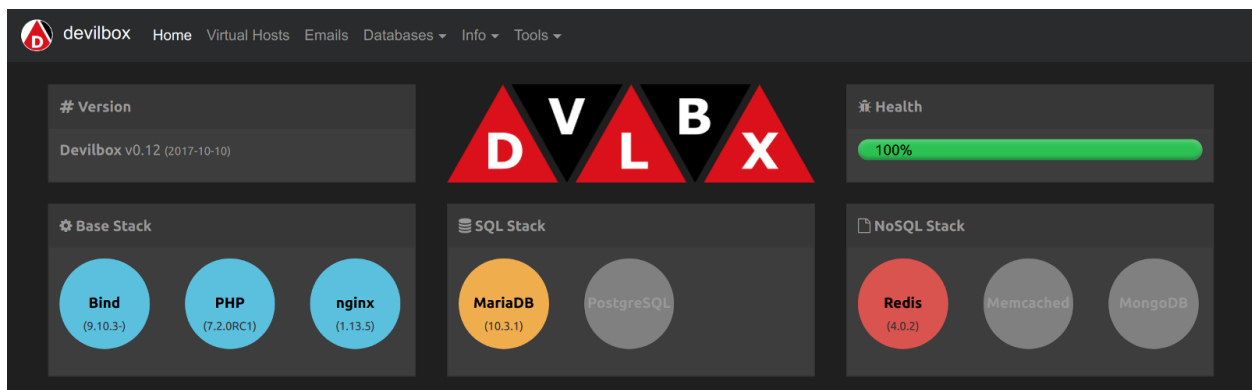


Fig. 2: Devilbox intranet: index dash view for some started container

---

**Important:**

*Find Docker Toolbox IP address* When you are using Docker Toolbox the Devilbox web server port will not be available on your host computer. You first have to find out on which IP address the Docker Toolbox machine is serving and use this one instead.

---

## 6.5 Checklist

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

**See also:**

*Troubleshooting*

# CHAPTER 7

---

## Devilbox intranet

---

The Devilbox 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*
  - *Adminer*
  - *phpMyAdmin*
  - *phpPgAdmin*
  - *phpRedMin*
  - *OpcacheGUI*
- *Settings*
  - *Password protect the intranet*
  - *Disable the intranet*
- *Checklist*

## 7.1 Devilbox tools

### 7.1.1 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`.

### 7.1.2 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.

### 7.1.3 Emails

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

### 7.1.4 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:

### 7.1.5 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:

## 7.2 Third-party tools


### 7.2.1 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.

### 7.2.2 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.




[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

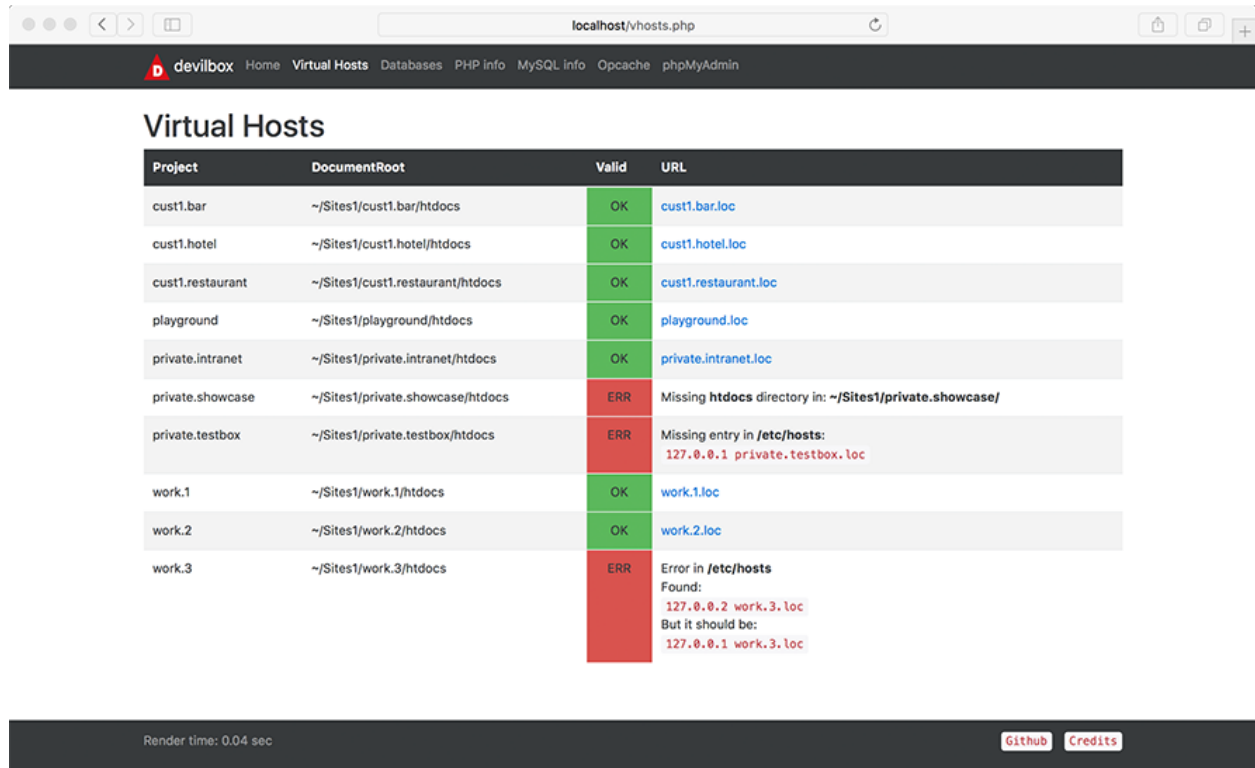
Docker	Host path	Docker path
php	/log/php-fpm-7.0	/var/log/php
httpd	/log/nginx-stable	/var/log/nginx-stable
mysql	/log/mariadb-10.1	/var/log/mysql
pgsql	/log/pgsql-9.6	/var/log/postgresql
redis	/log/redis-3.2	/var/log/redis
memcached	/log/memcached-1.4.21	/var/log/memcached
bind	-	-

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

7.2. Third-party tools

29

Fig. 1: Devilbox intranet: homepage



The screenshot shows a web browser at localhost/vhosts.php. The page title is "Virtual Hosts". It contains a table with the following data:

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 <b>htdocs</b> directory in: ~/Sites1/private.showcase/
private.testbox	~/Sites1/private.testbox/htdocs	ERR	Missing entry in <b>/etc/hosts</b> : 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 <b>/etc/hosts</b> Found: 127.0.0.2 work.3.loc But it should be: 127.0.0.1 work.3.loc

At the bottom of the page, it says "Render time: 0.04 sec" and has links for "Github" and "Credits".

Fig. 2: Devilbox intranet: available virtual hosts

### 7.2.3 phpPgAdmin

is a web-based administration tool for PostgreSQL. It is perfect for PostgreSQL DBAs, newbies, and hosting services.

### 7.2.4 phpRedMin

is a simple web interface to manage and monitor your Redis.

### 7.2.5 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).

## 7.3 Settings

### 7.3.1 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:

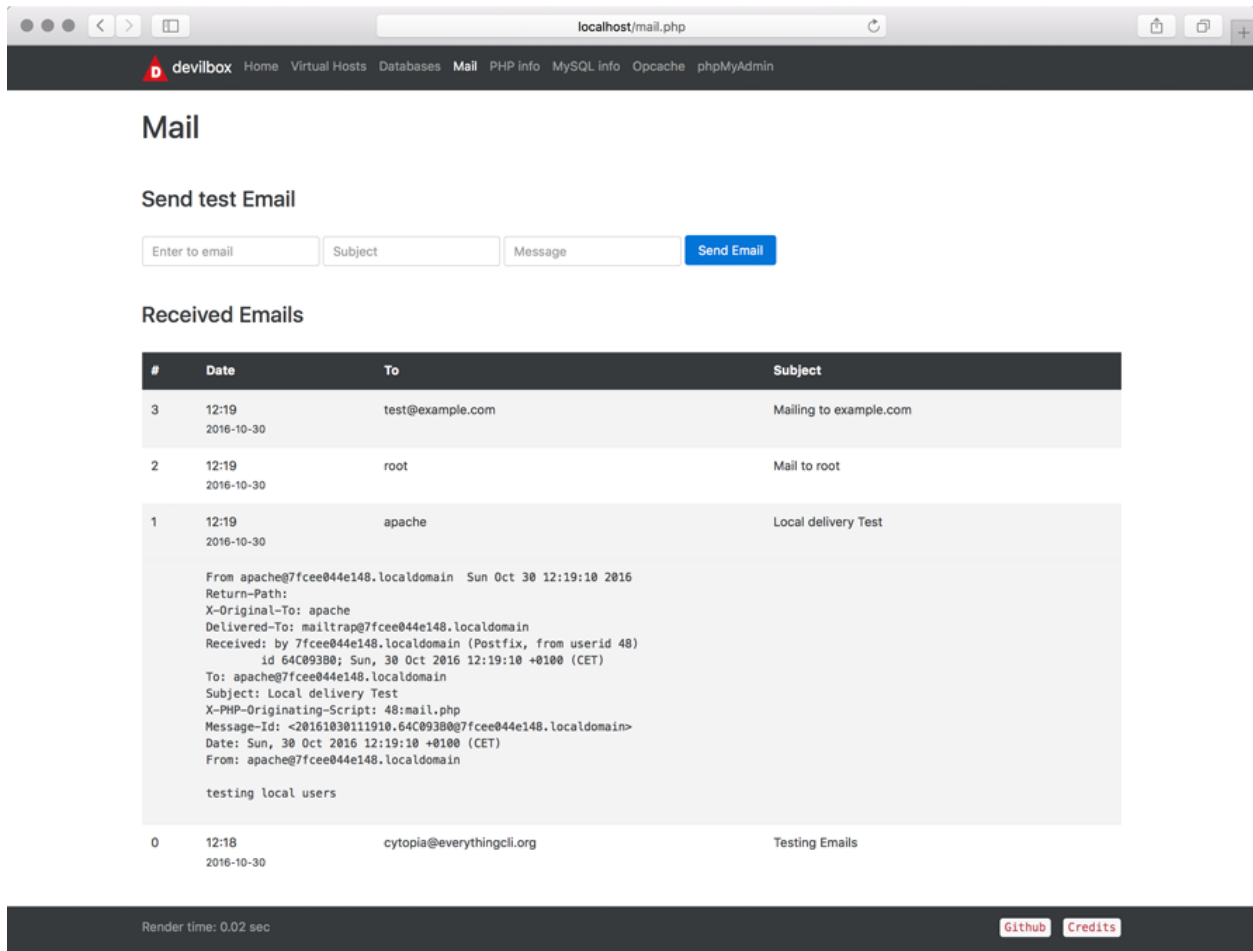
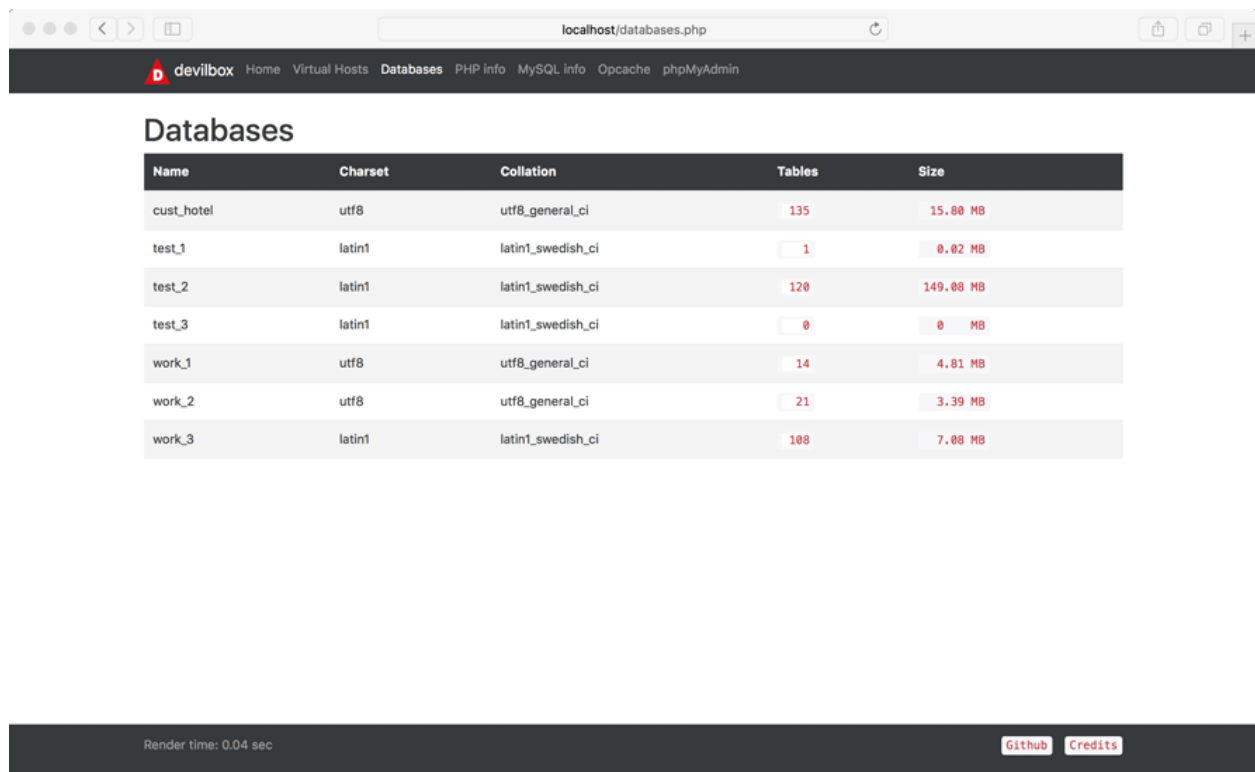


Fig. 3: Devilbox intranet: email catch-all overview



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

Render time: 0.04 sec

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

Fig. 4: Devilbox intranet: MySQL database overview

- *DEVILBOX\_UI\_PROTECT*
- *DEVILBOX\_UI\_PASSWORD*

### 7.3.2 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*

## 7.4 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

**See also:**

*Troubleshooting*

Fig. 5: Devilbox intranet: php info

devilbox

[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

Fig. 6: Devilbox intranet: MySQL info overview

# CHAPTER 8

---

## 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*

## 8.1 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
```

## 8.2 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.

## 8.3 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
```

## 8.4 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
```

## 8.5 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.

## 8.6 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 9

---

### 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*
- *Step 5: visit your project*
- *Step 6: create a hello world file*
- *Checklist*
- *Further examples*

### 9.1 Step 1: visit Intranet vhost page

Before starting, have a look at the vhost page at <http://localhost/vhosts.php> or <http://127.0.0.1/vhosts.php>

**See also:**

*Find Docker Toolbox IP address*

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

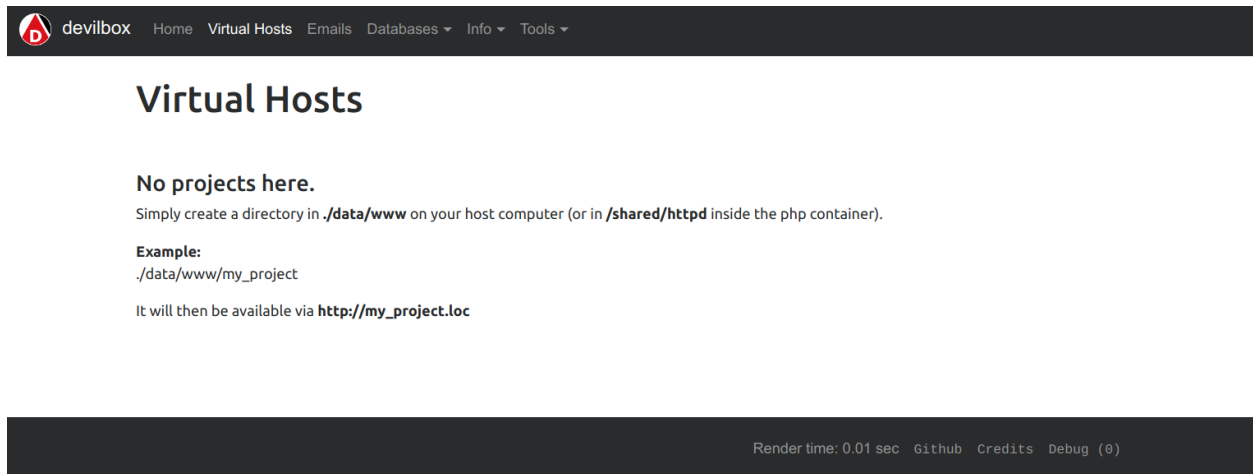


Fig. 1: Devilbox intranet: no projects created

## 9.2 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>

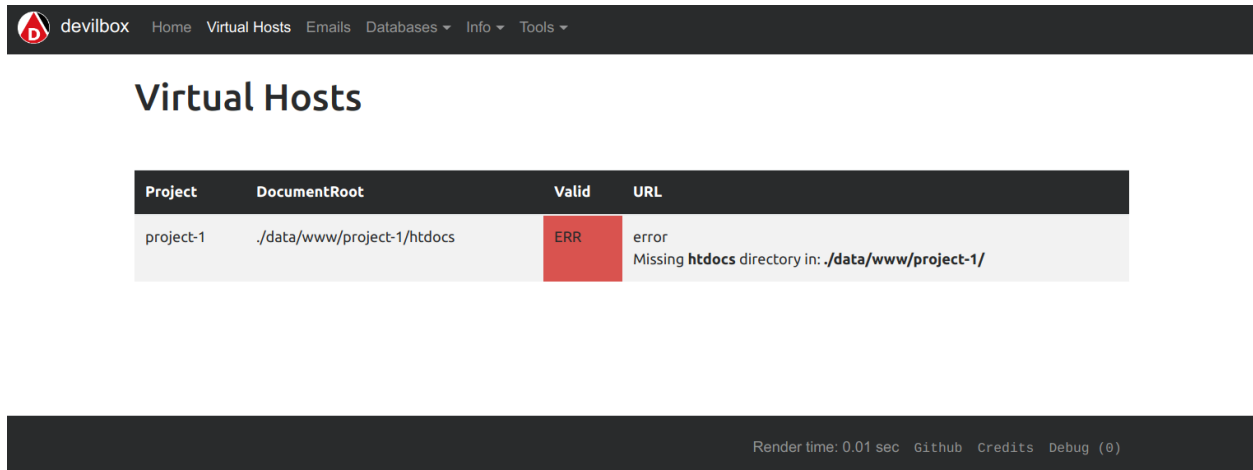
### So what has happened?

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.

## 9.3 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.



The screenshot shows the Devilbox interface with the 'Virtual Hosts' tab selected. A table lists the configuration for 'project-1'. The 'Valid' column shows an 'ERR' status, and the 'URL' column displays an error message: 'error Missing htdocs directory in: ./data/www/project-1/'.

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

Render time: 0.01 sec   Github   Credits   Debug (0)

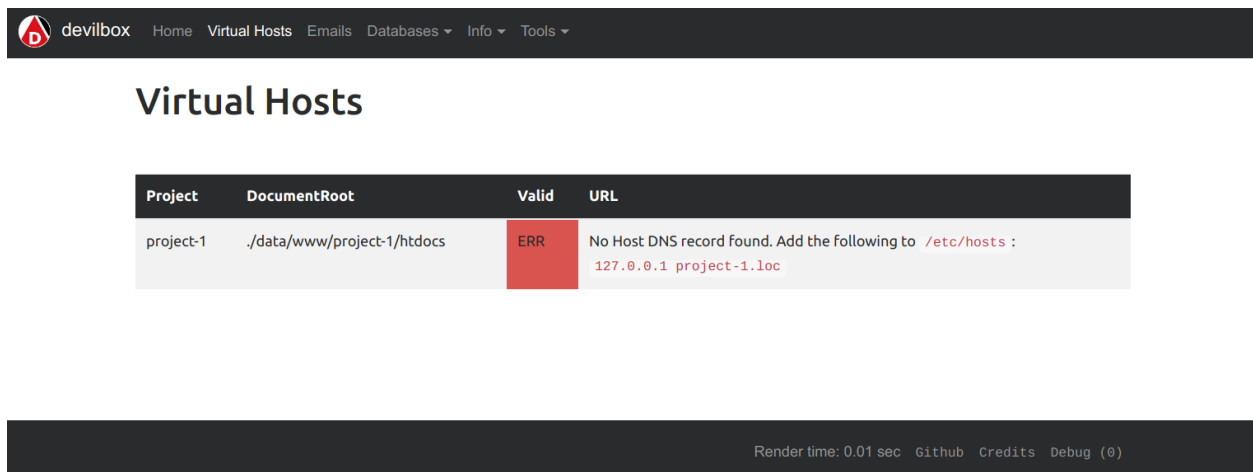
Fig. 2: Devilbox intranet: misssing htdocs directory

```
# 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>



The screenshot shows the Devilbox interface with the 'Virtual Hosts' tab selected. The table now shows a different error: 'No Host DNS record found. Add the following to /etc/hosts : 127.0.0.1 project-1.loc'.

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

Render time: 0.01 sec   Github   Credits   Debug (0)

Fig. 3: Devilbox intranet: misssing dns record

### 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` (or `C:\Windows\System32\drivers\etc` for Windows) file on your host operating system to solve this issue.

## 9.4 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*.

When using native Docker, the Devilbox intranet will provide you the exact string you need to paste into your `/etc/hosts` (or `C:\Windows\System32\drivers\etc` for Windows).

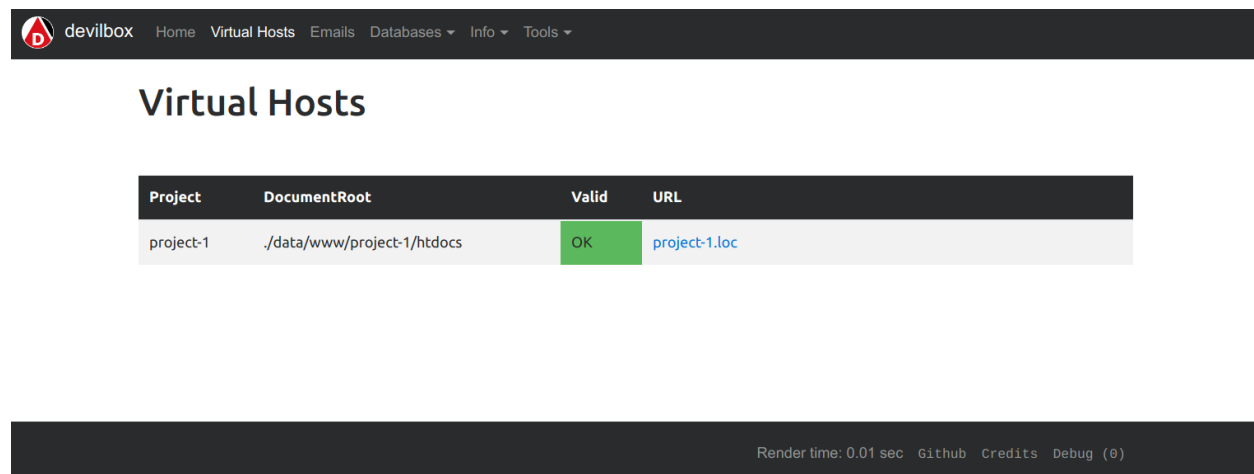
```
# Open your /etc/hosts file with sudo or root privileges
# and add the following DNS entry
host> sudo vi /etc/hosts

127.0.0.1 project-1.loc
```

See also:

- [Add project hosts entry on MacOS](#)
- [Add project hosts entry on Windows](#)

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



Project	DocumentRoot	Valid	URL
project-1	./data/www/project-1/htdocs	OK	<a href="http://project-1.loc">project-1.loc</a>

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

Fig. 4: Devilbox intranet: vhost setup successfully

### 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.

## 9.5 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>

### 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.

## 403 Forbidden

nginx/1.12.1

Fig. 5: Devilbox project: missing `index.php` or `index.html`

## 9.6 Step 6: create a hello world file

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

Fig. 6: Devilbox project: hello world on `index.php`

## 9.7 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

**See also:**

*Troubleshooting*

## 9.8 Further examples

If you already want to know how to setup specific frameworks on the Devilbox, jump directly to their articles:

**See also:**

**Well tested frameworks on the Devilbox**

- *Setup CakePHP*
- *Setup CodeIgniter*
- *Setup CraftCMS*
- *Setup Drupal*
- *Setup Joomla*
- *Setup Laravel*
- *Setup Magento 2*
- *Setup Phalcon*
- *Setup Photon CMS*
- *Setup PrestaShop*
- *Setup Shopware*
- *Setup Symfony*
- *Setup Typo3*
- *Setup Wordpress*
- *Setup Yii*
- *Setup Zend*

**See also:**

**Generic information for all unlisted frameworks**

- *Setup other Frameworks*



# 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 PHP 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

**See also:**

*Troubleshooting*



---

## 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

- *Implications*
  - *Configuration changes*
  - *Data changes*
- *Examples*
  - *Change PHP version*
    - \* *Stop the Devilbox*
    - \* *Edit the `.env` file*
    - \* *Start the Devilbox*
  - *Change web server version*
    - \* *Stop the Devilbox*
    - \* *Edit the `.env` file*
    - \* *Start the Devilbox*
  - *Change whatever version*
- *Gotchas*
- *Checklist*

## 11.1 Implications

### 11.1.1 Configuration changes

Be aware that every version has its own configuration files in the `cfg/` directory. If you switch to a different version, you might end up with a different custom configuration. This however only applies, if you have already customized the configuration for your current service.

See also:

- *php.ini*
- *php-fpm.conf*
- *apache.conf*
- *nginx.conf*
- *my.cnf*

### 11.1.2 Data changes

You also have to be aware that all database services (e.g.: MySQL, PostgreSQL, MongoDB, etc) use a per version data directory. If you change the database version you might find that you have an empty database when starting another version.

This is simply a pre-caution to prevent newer versions from upgrading the database files and accidentally making them incompatible for older versions.

If you want to take your data along, do a backup before switching the version and then re-import after the switch:

See also:

- *Backup and restore MySQL*
- *Backup and restore PostgreSQL*
- *Backup and restore MongoDB*

## 11.2 Examples

### 11.2.1 Change PHP version

#### Stop the Devilbox

Shut down the Devilbox in case it is still running:

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

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

## 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=5.2
#PHP_SERVER=5.3
#PHP_SERVER=5.4
#PHP_SERVER=5.5
#PHP_SERVER=5.6
#PHP_SERVER=7.0
PHP_SERVER=7.1
#PHP_SERVER=7.2
#PHP_SERVER=7.3
#PHP_SERVER=7.4
```

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=5.2
#PHP_SERVER=5.3
#PHP_SERVER=5.4
PHP_SERVER=5.5
#PHP_SERVER=5.6
#PHP_SERVER=7.0
#PHP_SERVER=7.1
#PHP_SERVER=7.2
#PHP_SERVER=7.3
#PHP_SERVER=7.4
```

## Start the Devilbox

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

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

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

### See also:

*Start the Devilbox*

## 11.2.2 Change web server version

### Stop the Devilbox

Shut down the Devilbox in case it is still running:

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

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

### Edit the `.env` file

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

Listing 3: `.env`

```
#HTTPD_SERVER=apache-2.2
#HTTPD_SERVER=apache-2.4
HTTPD_SERVER=nginx-stable
#HTTPD_SERVER=nginx-mainline
```

As you can see, all available values are already there, but commented. Only one is uncommented. In this example it is `nginx-stable`, which is the web server 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 Apache 2.2, you would change the `.env` file like this:

Listing 4: `.env`

```
HTTPD_SERVER=apache-2.2
#HTTPD_SERVER=apache-2.4
#HTTPD_SERVER=nginx-stable
#HTTPD_SERVER=nginx-mainline
```

### Start the Devilbox

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

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

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

### See also:

*Start the Devilbox*



### 11.2.3 Change whatever version

When you have read how to change the PHP or web server 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`

## 11.3 Gotchas

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

Consider this `.env` file:

Listing 5: `.env`

```
#PHP_SERVER=5.2
#PHP_SERVER=5.3
#PHP_SERVER=5.4
PHP_SERVER=5.5
#PHP_SERVER=5.6
PHP_SERVER=7.0
#PHP_SERVER=7.1
#PHP_SERVER=7.2
#PHP_SERVER=7.3
#PHP_SERVER=7.4
```

Both, PHP 5.5 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.

## 11.4 Checklist

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

**See also:**

*Troubleshooting*



# CHAPTER 12

## Setup Auto DNS

If you don't want to add host 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*
  - *Docker on Linux*
  - *Docker for Mac*
  - *Docker for Windows*
- *Docker Toolbox*
  - *Prerequisites*
  - *Actual setup*

## 12.1 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

### 12.1.1 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
```

### 12.1.2 Docker on Linux

Your DNS server IP address is 127.0.0.1.

**See also:**

*Add custom DNS server on Linux*

### 12.1.3 Docker for Mac

Your DNS server IP address is 127.0.0.1.

**See also:**

*Add custom DNS server on MacOS*

### 12.1.4 Docker for Windows

Your DNS server IP address is 127.0.0.1.

**See also:**

*Add custom DNS server on Windows*

## 12.2 Docker Toolbox

**See also:**

*Docker Toolbox and the Devilbox*

This part applies equally for Docker Toolbox on MacOS and on Windows:

### 12.2.1 Prerequisites

- *LOCAL\_LISTEN\_ADDR* must be empty in order to listen on all interfaces
- *HOST\_PORT\_BIND* must be set to 53

You need to create three port-forwards to make the DNS and web server available on your host os:

- Port 80 from the Docker Toolbox virtual machine must be port-forwarded to 127.0.0.1:80 on your host os
- Port 443 from the Docker Toolbox virtual machine must be port-forwarded to 127.0.0.1:443 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

Assuming the Docker Toolbox IP is 192.168.99.100 your forwards must be as follows:

From IP	From port	To IP	To port
192.168.99.100	53	127.0.0.1	53
192.168.99.100	80	127.0.0.1	80
192.168.99.100	443	127.0.0.1	443

**See also:**

- *SSH port-forward on Docker Toolbox from host*
- *Find Docker Toolbox IP address*

### 12.2.2 Actual setup

---

**Important:** After settings this up, follow the above guides for **Docker for Mac** or **Docker for Windows** to finish the setup.

---



---

### Setup valid HTTPS

---

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.

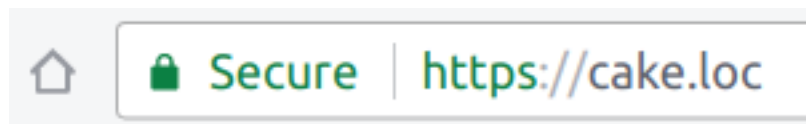


Fig. 1: Valid HTTPS will automatically be available for all projects

#### Table of Contents

- *TL;DR*
- *How does it work*
  - *Certificate Authority*
  - *SSL Certificates*
- *Import the CA into your browser*
  - *Chrome / Chromium*
  - *Firefox*
- *Further Reading*

### 13.1 TL;DR

Import the Certificate Authority into your browser and you are all set.

## 13.2 How does it work

### 13.2.1 Certificate Authority

When the Devilbox starts up for the first time, it will generate a 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
```

### 13.2.2 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 *Setup 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.

## 13.3 Import the CA into your browser

---

**Important:** Importing the CA into the browser is also recommended and required for the Devilbox intranet page to work properly. You may also import the CA into your Operating System's Keystore. Information on that is available at .

---

### 13.3.1 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.



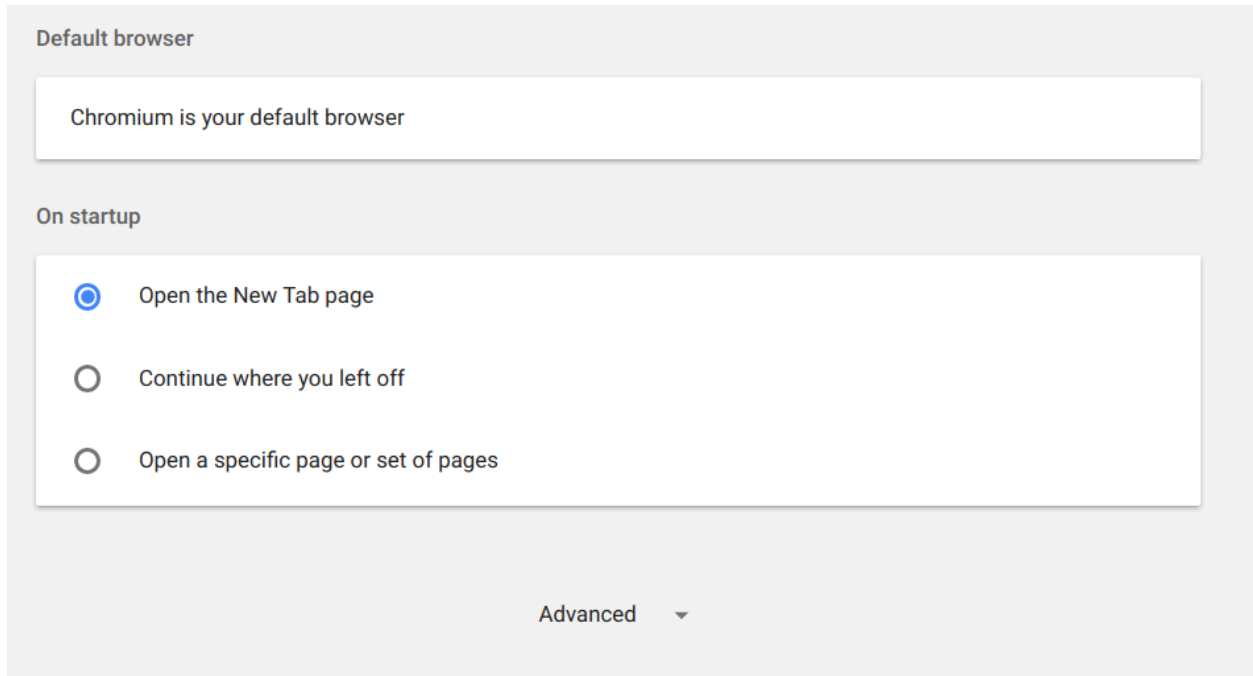


Fig. 2: Click on Advanced

### 13.3.2 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 imported 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.

## 13.4 Further Reading

See also:

.env variable: `DEVILBOX_UI_SSL_CN`

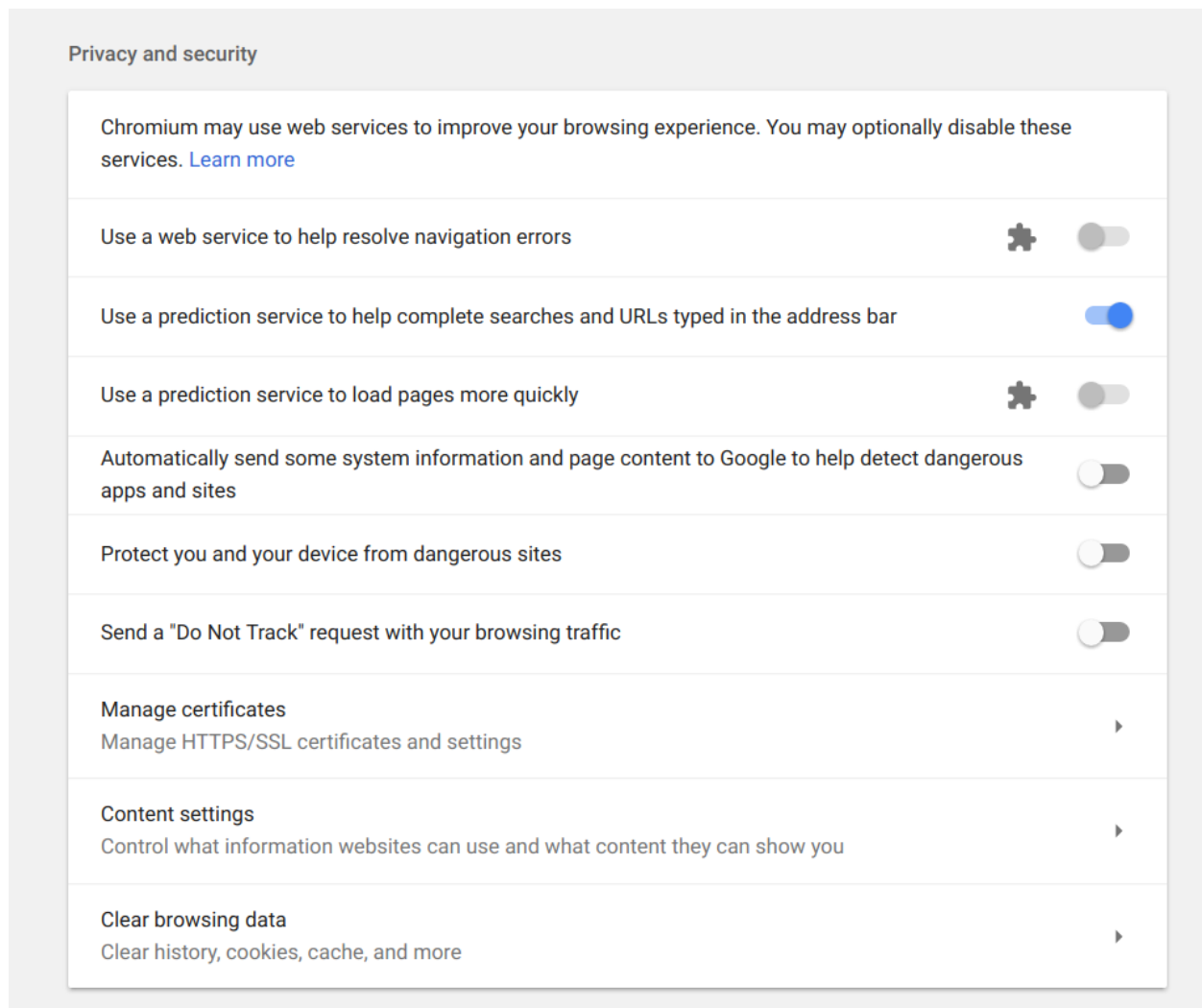


Fig. 3: Click on Manage certificates

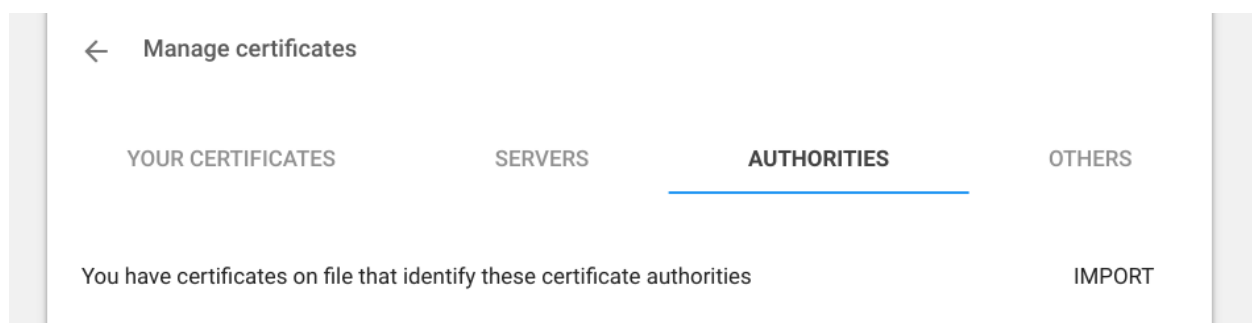


Fig. 4: Click on IMPORT in the AUTHORITIES tab

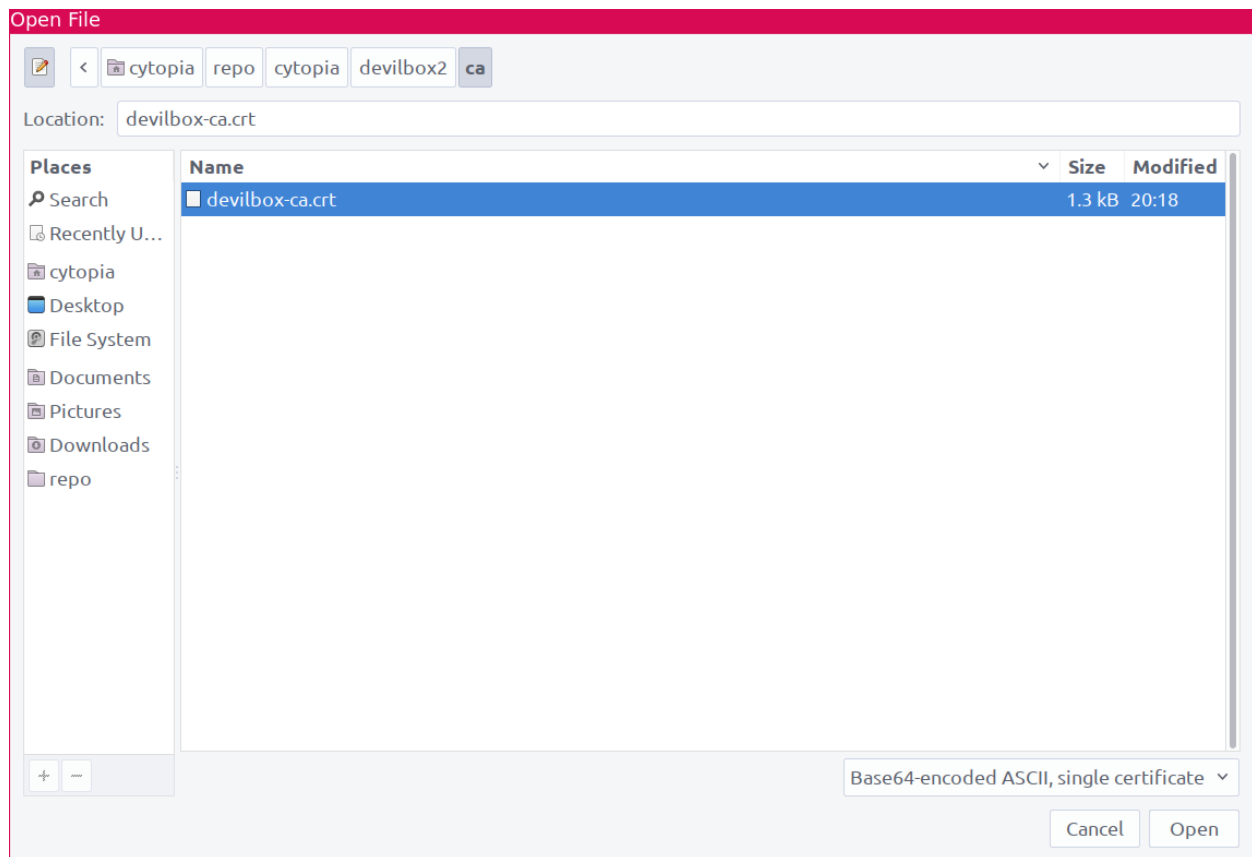


Fig. 5: **Note:** your file manager might look different

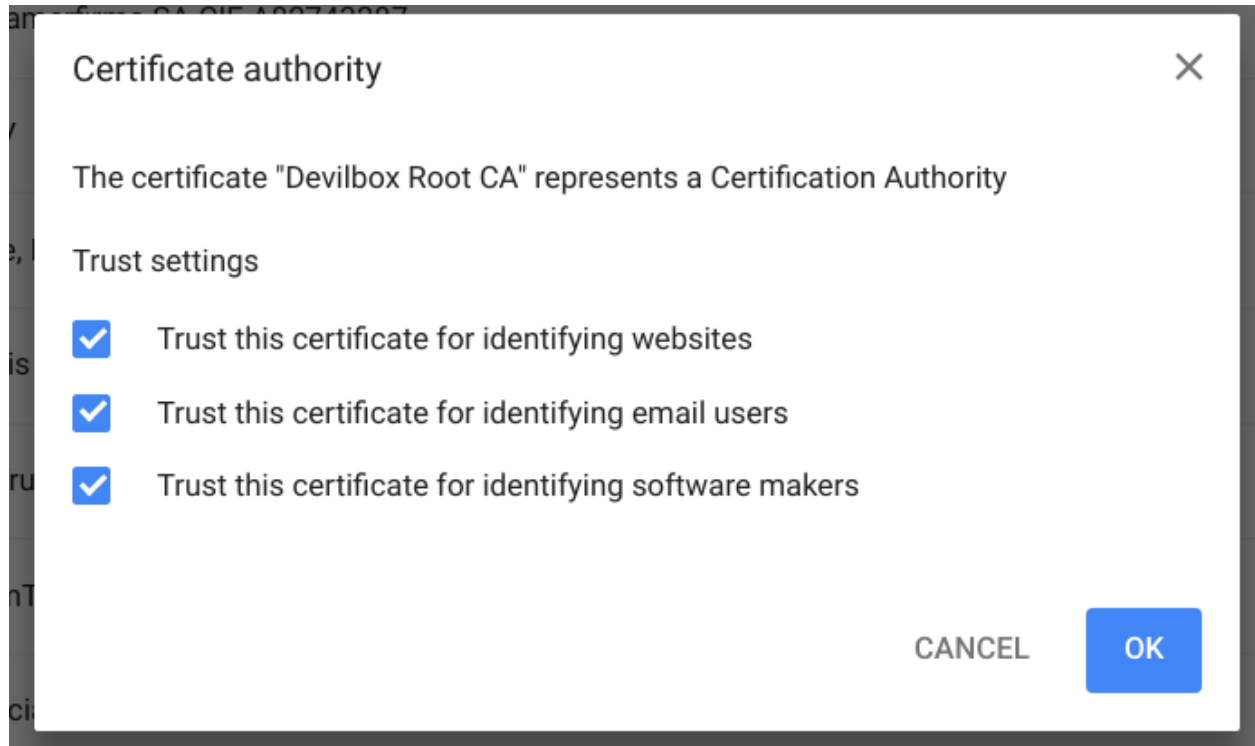


Fig. 6: Tell Chrome to trust this CA

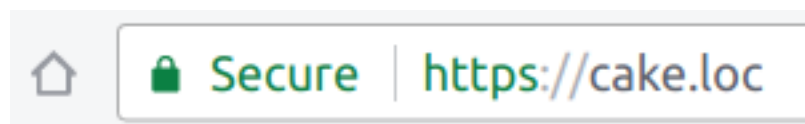


Fig. 7: Valid HTTPS will automatically be available for all projects

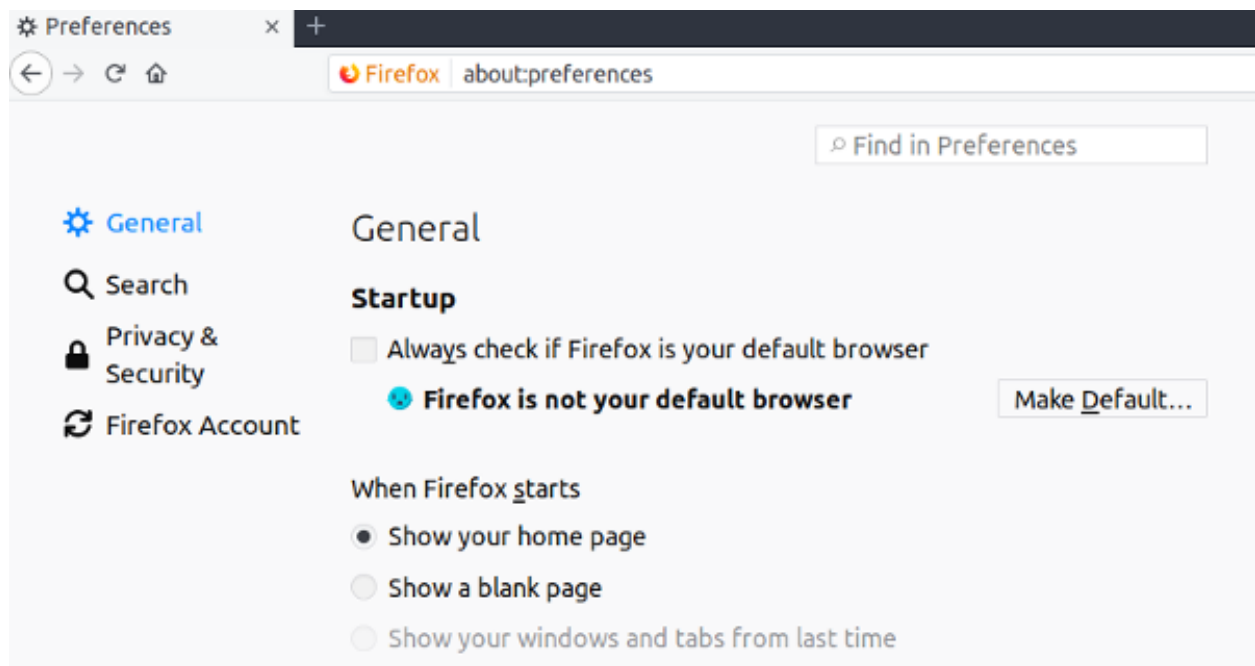


Fig. 8: Click on Privacy & Security in the left menu bar

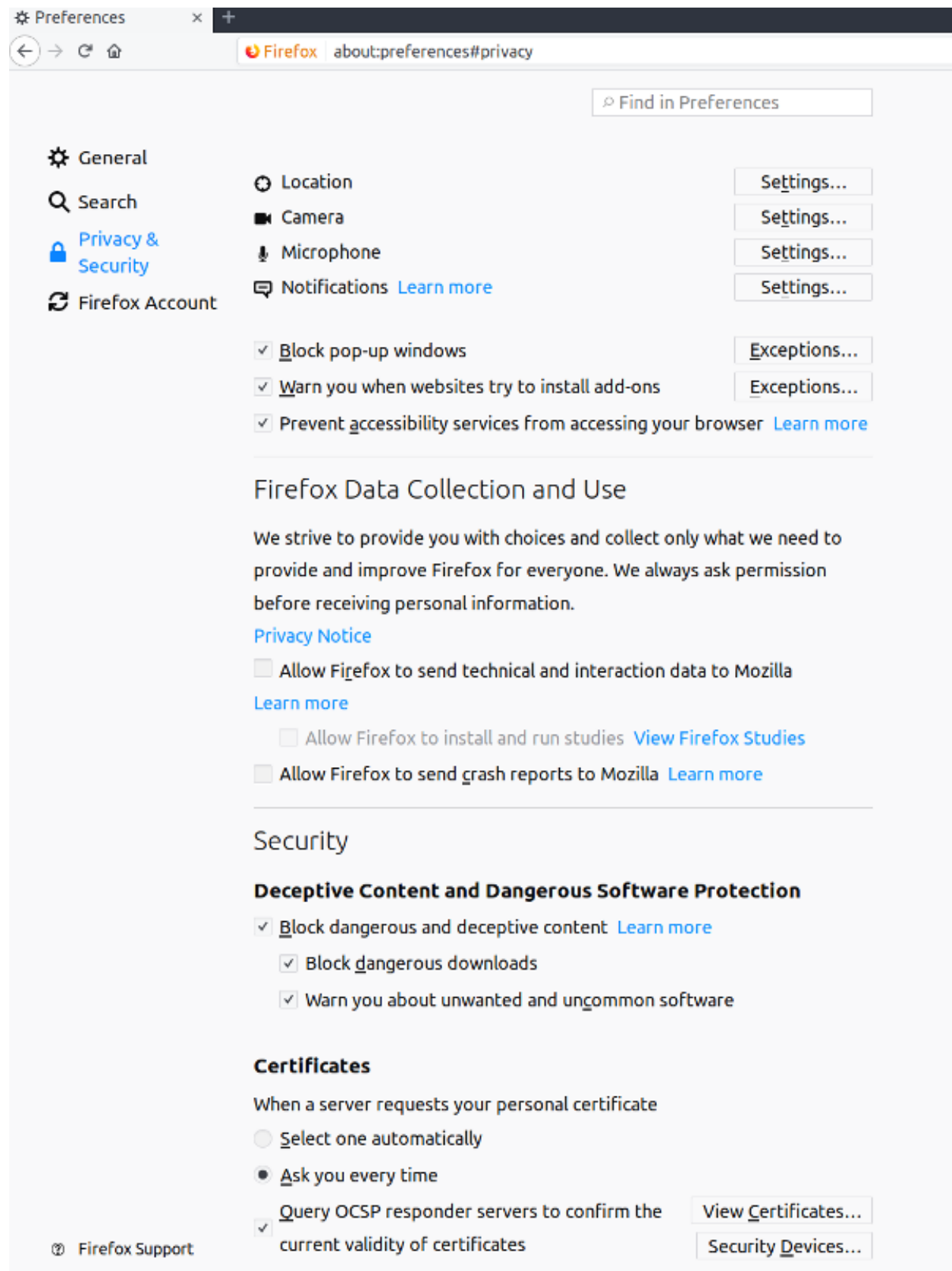


Fig. 9: Click on View Certificates

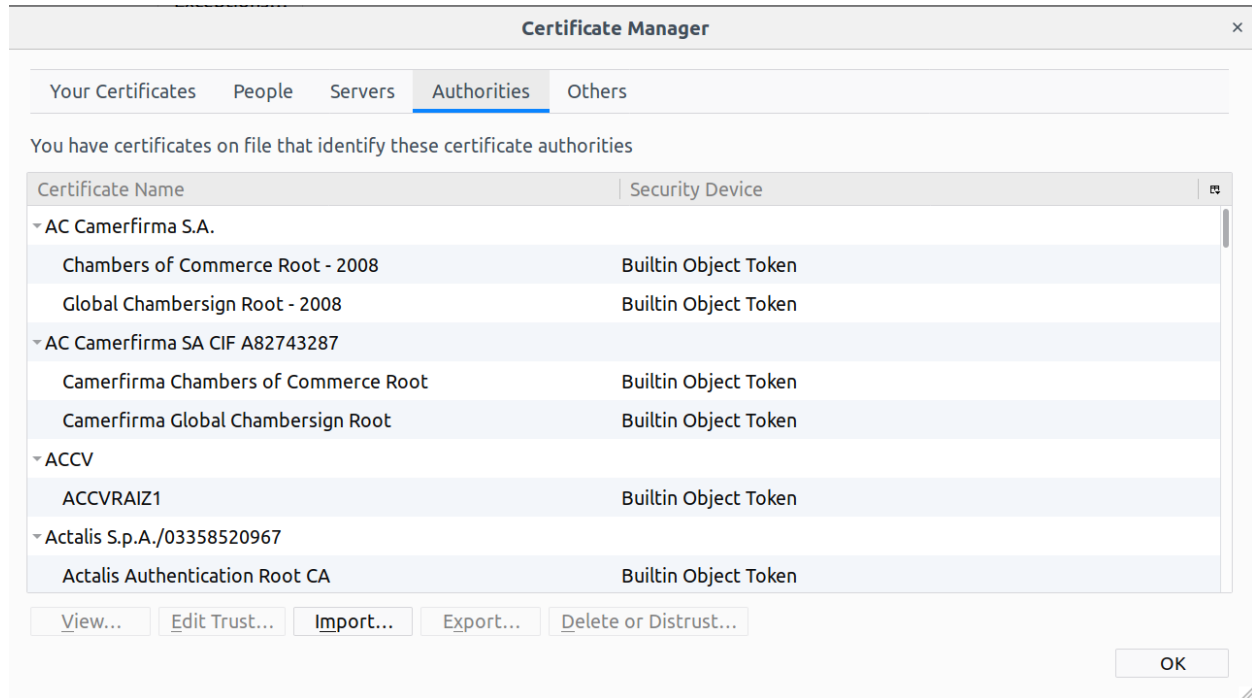
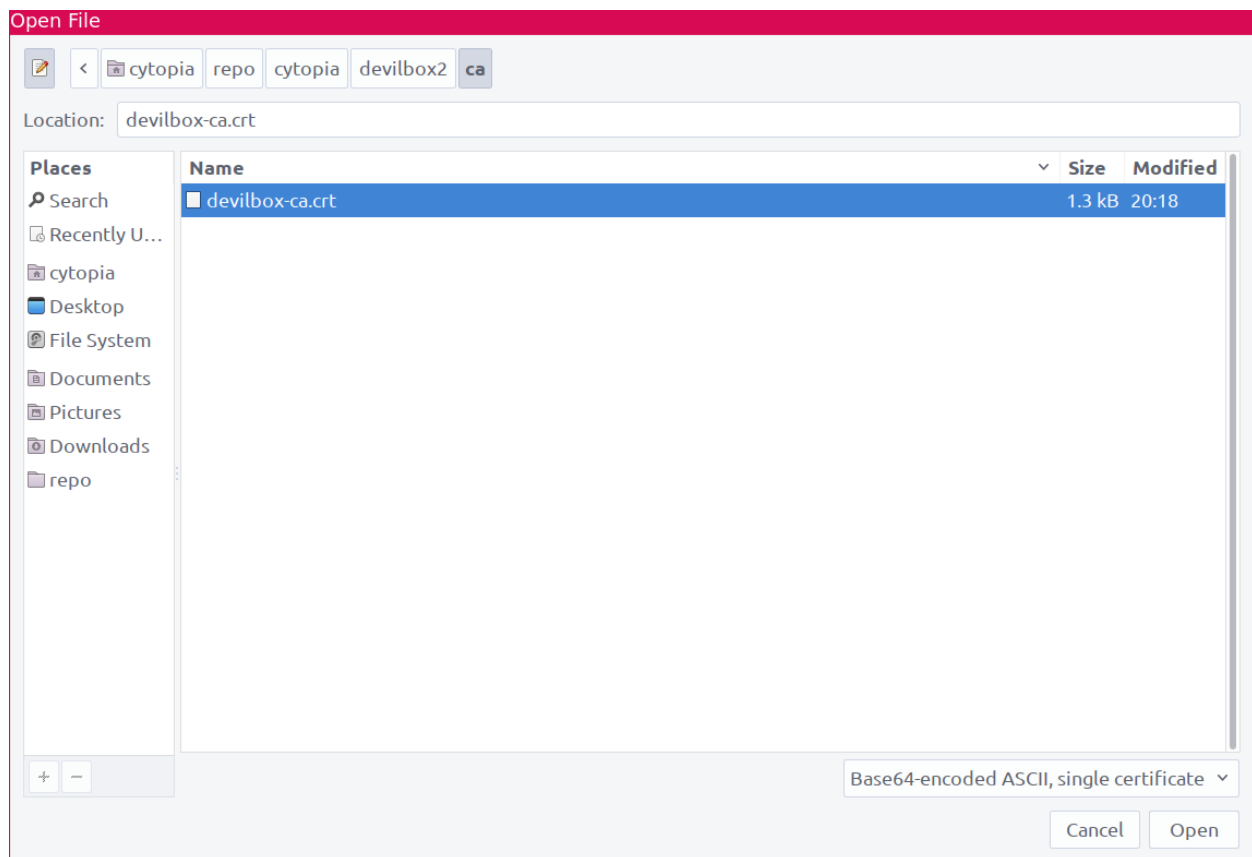


Fig. 10: Click on Import in the Authorities tab

Fig. 11: **Note:** your file manager might look different

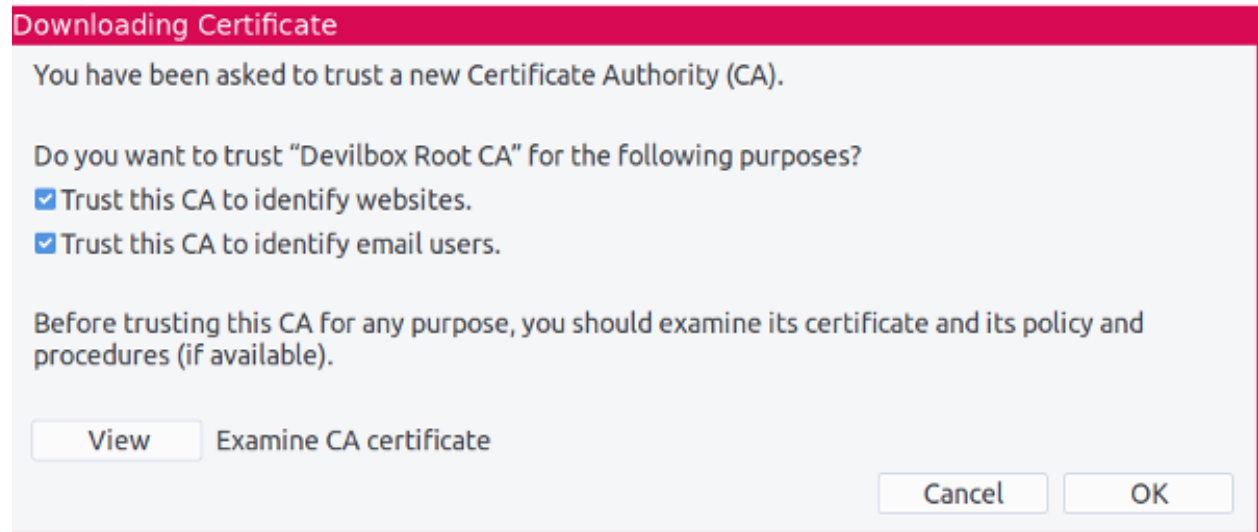


Fig. 12: Tell Firefox to trust this CA

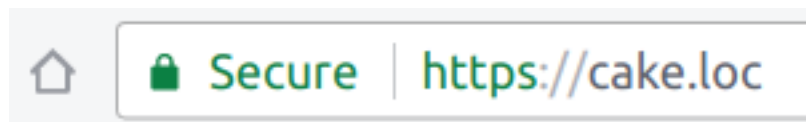


Fig. 13: Valid HTTPS will automatically be available for all projects



---

## Configure PHP Xdebug

---

This section explains in depth how to enable and use PHP Xdebug with the Devilbox.

### Table of Contents

- *Introduction*
- *Configure PHP container for Xdebug*
- *Configure your IDE/editor for Xdebug*
  - *Path mapping*
  - *IDE key*
  - *Configuration*

## 14.1 Introduction

In order to have a working Xdebug, you need to ensure two things:

1. **PHP Xdebug** must be configured and enabled in PHP itself
2. Your **IDE/editor** must be configured and requires a way talk to PHP

Configuring PHP Xdebug will slightly differ when configuring it for a dockerized environment. This is due to the fact that Docker versions on different host os have varying implementations of how they connect back to the host.

Most IDE or editors will also require different configurations for how they talk to PHP Xdebug. This is at least most likely the case for `xdebug.idekey`.

## 14.2 Configure PHP container for Xdebug

### 14.2.1 Xdebug options explained

#### Table of Contents

- *Example*
  - *default\_enable*
  - *remote\_enable*
  - *remote\_handler*
  - *remote\_port*
  - *remote\_autostart*
  - *idekey*
  - *remote\_log*

#### Example

Consider the following `xdebug.ini` as an example:

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:

##### **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 ). 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.

---

**Important:** Many IDE/editors require a specific value for `xdebug.idekey`. Make sure you pay special attention to that variable when it comes to configuring your IDE/editor.

---

### 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.

## 14.2.2 Configure Xdebug: Docker Toolbox

Docker Toolbox regardless of running on MacOS or Windows requires an additional port-forward from your host operating system to the Docker Toolbox machine.

### Table of Contents

- [\*Prerequisites\*](#)
- [\*Configure php.ini\*](#)

### Prerequisites

Ensure you know how to customize `php.ini` values for the Devilbox.

#### See also:

- [\*php.ini\*](#)
- [\*Xdebug options explained\*](#)

## Configure php.ini

The following example show how to configure PHP Xdebug for PHP 5.6:

### 1. Forward host port 9000 to Docker Toolbox machine

Your IDE/editor will open up port 9000 on your host operating system. PHP Xdebug requires this port to connect to in order to send Xdebug events. As Docker Toolbox itself runs in a virtual machine, you need to forward traffic from the same port in that virtual machine back to your host operating system.

See also:

- *SSH port-forward on Docker Toolbox from host*
- *SSH port-forward on host to Docker Toolbox*

### 2. Create xdebug.ini

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

# Navigate to PHP 5.6 ini configuration directory
host> cd cfg/php-ini-5.6/

# Create and open debug.ini file
host> vi xdebug.ini
```

### 3. Paste the following content into xdebug.ini

Once the por-forward is up, the configuration matches the one for Docker on Linux

See also:

- CNAME for *Docker on Linux*

Listing 2: xdebug.ini

```
; Defaults
xdebug.remote_enable=1
xdebug.remote_port=9000

; The Docker Toolbox way
xdebug.remote_connect_back=0
xdebug.remote_host=docker.for.lin.host.internal

; idekey value is specific to each editor
; Verify with your IDE/editor documentation
xdebug.idekey=PHPSTORM

; Optional: Set to true to auto-start xdebug
xdebug.remote_autostart=false
```

### 4. Configure your IDE/editor

See also:

- *Configure Xdebug for Atom*
- *Configure Xdebug for PhpStorm*
- *Configure Xdebug for Sublime Text 3*
- *Configure Xdebug for Visual Studio Code*

---

**Important:** Depending on your IDE/editor, you might have to adjust `xdebug.idekey` in the above configured `xdebug.ini`.

---

## 5. Restart the Devilbox

Restarting the Devilbox is important in order for it to read the new PHP settings.

## 14.2.3 Configure Xdebug: Docker on Linux

Docker on Linux allows Xdebug to automatically connect back to the host system without the need of an explicit IP address.

### Table of Contents

- *Prerequisites*
- *Configure php.ini*

### Prerequisites

Ensure you know how to customize `php.ini` values for the Devilbox.

See also:

- *php.ini*
- *Xdebug options explained*

### Configure php.ini

Configuring Xdebug for Docker on Linux is straight forward and you must only pay attention to two variables highlighted below.

The following example show how to configure PHP Xdebug for PHP 5.6:

#### 1. Create `xdebug.ini`

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

# Navigate to PHP 5.6 ini configuration directory
host> cd cfg/php-ini-5.6/

# Create and open debug.ini file
host> vi xdebug.ini
```

#### 2. Paste the following content into `xdebug.ini`

Listing 3: `xdebug.ini`

```
; Defaults
xdebug.default_enable=1
xdebug.remote_enable=1
```

(continues on next page)

(continued from previous page)

```
xdebug.remote_port=9000

; The Linux way
xdebug.remote_connect_back=1

; idekey value is specific to each editor
; Verify with your IDE/editor documentation
xdebug.idekey=PHPSTORM

; Optional: Set to true to auto-start xdebug
xdebug.remote_autostart=false
```

### 3. Configure your IDE/editor

**See also:**

- *Configure Xdebug for Atom*
- *Configure Xdebug for PhpStorm*
- *Configure Xdebug for Sublime Text 3*
- *Configure Xdebug for Visual Studio Code*

---

**Important:** Depending on your IDE/editor, you might have to adjust `xdebug.idekey` in the above configured `xdebug.ini`.

---

### 4. Restart the Devilbox

Restarting the Devilbox is important in order for it to read the new PHP settings.

## 14.2.4 Configure Xdebug: Docker for Mac

Docker for MacOS requires a well known IP address in order to connect to the host operating system. This can be achieved with two different approaches described below.

**See also:**

<https://forums.docker.com/t/ip-address-for-xdebug/10460/32>

#### Table of Contents

- *Prerequisites*
- *Configure php.ini: CNAME alias*
- *Configure php.ini: Host alias*

### Prerequisites

Ensure you know how to customize `php.ini` values for the Devilbox and Xdebug is enabled.

**See also:**

- *php.ini*

- *Xdebug options explained*

## Configure php.ini: CNAME alias

**Option 1:** This option is the easiest to setup, but was also very fragile on many Docker versions.

Docker for Mac received many default CNAMEs throughout its versions. The most recent and active one is `host.docker.internal`. Use this CNAME as the remote address for Xdebug to connect to your IDE/editor on your host os.

**See also:**

**CNAME for *Docker for Mac*** In case `host.docker.internal` is not resolvable, read on here for alternative CNAME's on Docker for Mac

**Important:** Before you try this approach, verify that the PHP Docker container is actually able to resolve `host.docker.internal`:

```
host> cd path/to/devilbox
host> ./shell.sh
php> ping host.docker.internal
```

In case it is not resolvable, stick to the host alias address approach.

The following example show how to configure PHP Xdebug for PHP 5.6:

### 1. Create xdebug.ini

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

# Navigate to PHP 5.6 ini configuration directory
host> cd cfg/php-ini-5.6/

# Create and open debug.ini file
host> vi xdebug.ini
```

### 2. Paste the following content into xdebug.ini

Listing 4: xdebug.ini

```
; Defaults
xdebug.default_enable=1
xdebug.remote_enable=1
xdebug.remote_port=9000

; The MacOS way (CNAME)
xdebug.remote_connect_back=0
xdebug.remote_host=host.docker.internal

; idekey value is specific to each editor
; Verify IDE/editor documentation
xdebug.idekey=PHPSTORM

; Optional: Set to true to auto-start xdebug
xdebug.remote_autostart=false
```

### 3. Configure your IDE/editor

See also:

- *Configure Xdebug for Atom*
- *Configure Xdebug for PhpStorm*
- *Configure Xdebug for Sublime Text 3*
- *Configure Xdebug for Visual Studio Code*

---

**Important:** Depending on your IDE/editor, you might have to adjust `xdebug.idekey` in the above configured `xdebug.ini`.

---

### 4. Restart the Devilbox

Restarting the Devilbox is important in order for it to read the new PHP settings.

#### Configure php.ini: Host alias

**Option 2:** This is the most general option that should work with any Docker version on MacOS, it does however require a few changes in your system.

---

**Important:** Ensure you have created an *Host address alias on MacOS* and `10.254.254.254` is aliased to your localhost.

---

The following example show how to configure PHP Xdebug for PHP 5.6:

#### 1. Create xdebug.ini

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

# Navigate to PHP 5.6 ini configuration directory
host> cd cfg/php-ini-5.6/

# Create and open debug.ini file
host> vi xdebug.ini
```

#### 2. Paste the following content into xdebug.ini

Listing 5: xdebug.ini

```
; Defaults
xdebug.default_enable=1
xdebug.remote_enable=1
xdebug.remote_port=9000

; The MacOS way (host alias)
xdebug.remote_connect_back=0
xdebug.remote_host=10.254.254.254

; idekey value is specific to each editor
; Verify with your IDE/editor documentation
xdebug.idekey=PHPSTORM
```

(continues on next page)



(continued from previous page)

```
; Optional: Set to true to auto-start xdebug
xdebug.remote_autostart=false
```

### 3. Configure your IDE/editor

See also:

- *Configure Xdebug for Atom*
- *Configure Xdebug for PhpStorm*
- *Configure Xdebug for Sublime Text 3*
- *Configure Xdebug for Visual Studio Code*

---

**Important:** Depending on your IDE/editor, you might have to adjust `xdebug.idekey` in the above configured `xdebug.ini`.

---

### 4. Restart the Devilbox

Restarting the Devilbox is important in order for it to read the new PHP settings.

## 14.2.5 Configure Xdebug: Docker for Windows

Docker for Windows requires a well known IP address in order to connect to the host operating system. This can be achieved with two different approaches described below.

### Table of Contents

- *Prerequisites*
- *Configure php.ini: CNAME alias*
- *Configure php.ini: Get IP manually*

### Prerequisites

Ensure you know how to customize `php.ini` values for the Devilbox and Xdebug is enabled.

See also:

- *php.ini*
- *Xdebug options explained*

### Configure php.ini: CNAME alias

**Option 1:** This option is the easiest to setup, but was also very fragile on many Docker versions.

Docker for Windows received many default CNAMEs throughout its versions. The most recent and active one is `host.docker.internal`. Use this CNAME as the remote address for Xdebug to connect to your IDE/editor on your host os.

See also:

**CNAME for *Docker for Windows*** In case `host.docker.internal` is not resolvable, read on here for alternative CNAME's on Docker for Windows

---

**Important:** Before you try this approach, verify that the PHP Docker container is actually able to resolve `host.docker.internal`:

```
C:\> cd path\to\devilbox
C:\> shell.bat
php> ping host.docker.internal
```

In case it is not resolvable, stick to the host alias address approach.

---

The following example show how to configure PHP Xdebug for PHP 5.6:

### 1. Create `xdebug.ini`

1. Navigate to the Devilbox directory
2. Navigate to `cfg\php-ini-5.6\` directory
3. Create a new file named `xdebug.ini`

---

**Important:** Pay attention that windows is not adding `.txt` as a file extension.

---

### 2. Paste the following content into `xdebug.ini`

Listing 6: `xdebug.ini`

```
; Defaults
xdebug.default_enable=1
xdebug.remote_enable=1
xdebug.remote_port=9000

; The Windows way (CNAME)
xdebug.remote_connect_back=0
xdebug.remote_host=host.docker.internal

; idekey value is specific to each editor
; Verify IDE/editor documentation
xdebug.idekey=PHPSTORM

; Optional: Set to true to auto-start xdebug
xdebug.remote_autostart=false
```

### 3. Configure your IDE/editor

See also:

- *Configure Xdebug for Atom*
- *Configure Xdebug for PhpStorm*
- *Configure Xdebug for Sublime Text 3*
- *Configure Xdebug for Visual Studio Code*

---

**Important:** Depending on your IDE/editor, you might have to adjust `xdebug.idekey` in the above configured `xdebug.ini`.

---

#### 4. Restart the Devilbox

Restarting the Devilbox is important in order for it to read the new PHP settings.

#### Configure php.ini: Get IP manually

**Option 2:** This is the most general option that should work with any Docker version on Windows. it does however require a small manual step.

The following example show how to configure PHP Xdebug for PHP 5.6:

##### 1. Gather IP address for Xdebug

On Windows you will have to manually gather the IP address and add it to `xdebug.remote_host`.

1. Open command line
2. Enter `ipconfig`
3. Look for the IP4 address in DockerNAT (e.g.: `192.168.246.1`)

**See also:**

*Open a terminal on Windows*

---

**Important:** `192.168.246.1` is meant as an example and will eventually differ on your system. Ensure you substitute it with the correct IP address.

---

##### 2. Create xdebug.ini

1. Navigate to the Devilbox directory
2. Navigate to `cfg\php-ini-5.6\` directory
3. Create a new file named `xdebug.ini`

---

**Important:** Pay attention that windows is not adding `.txt` as a file extension.

---

##### 3. Paste the following content into xdebug.ini

Listing 7: xdebug.ini

```
; Defaults
xdebug.default_enable=1
xdebug.remote_enable=1
xdebug.remote_port=9000

; The Windows way (IP address)
xdebug.remote_connect_back=0
xdebug.remote_host=192.168.246.1

; idekey value is specific to each editor
; Verify IDE/editor documentation
```

(continues on next page)

(continued from previous page)

```
xdebug.idekey=PHPSTORM

; Optional: Set to true to auto-start xdebug
xdebug.remote_autostart=false
```

---

**Important:** 192.168.246.1 is meant as an example and will eventually differ on your system. Ensure you substitute it with the correct IP address.

---

## 4. Configure your IDE/editor

See also:

- [\*Configure Xdebug for Atom\*](#)
- [\*Configure Xdebug for PhpStorm\*](#)
- [\*Configure Xdebug for Sublime Text 3\*](#)
- [\*Configure Xdebug for Visual Studio Code\*](#)

---

**Important:** Depending on your IDE/editor, you might have to adjust `xdebug.idekey` in the above configured `xdebug.ini`.

---

## 5. Restart the Devilbox

Restarting the Devilbox is important in order for it to read the new PHP settings.

The following gives you a step-by-step guide on how to setup PHP Xdebug for the Devilbox depending on what host operating system you are using.

Be reminded that PHP configuration is always done per version, i.e. having it configured for PHP 7.2, does not enable it for any other versions.

See also:

- [\*Xdebug options explained\*](#)
- [\*Configure Xdebug: Docker on Linux\*](#)
- [\*Configure Xdebug: Docker for Mac\*](#)
- [\*Configure Xdebug: Docker for Windows\*](#)
- [\*Configure Xdebug: Docker Toolbox\*](#) (Mac or Windows)

## 14.3 Configure your IDE/editor for Xdebug

After you have setup PHP Xdebug as referenced above, you can continue to configure your currently used IDE/editor. Most IDE/editors will usually be configured in a very similar way, which comes down to two main settings;

### 14.3.1 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 actually absolute path of it, when setting up your IDE.

---

### 14.3.2 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`.

#### Configure Xdebug for Atom

##### Table of Contents

- [Prerequisites](#)
- [Assumption](#)
- [Configuration](#)

#### Prerequisites

Ensure that `xdebug.idekey` is set to `xdebug.atom` in your PHP Xdebug configuration.

##### See also:

- [Configure Xdebug: Docker on Linux](#)
- [Configure Xdebug: Docker for Mac](#)
- [Configure Xdebug: Docker for Windows](#)
- [Configure Xdebug: Docker Toolbox](#)

#### Assumption

For the sake of this example, we will assume the following paths:

Directory	Path
Devilbox git directory	<code>/home/cytopia/repo/devilbox</code>
<code>HOST_PATH_HTTPD_DATADIR</code>	<code>./data/www</code>
Resulting local project path	<code>/home/cytopia/repo/devilbox/data/www</code>

The **Resulting local project path** is the path where all projects are stored locally on your host operating system. No matter what this path is, the equivalent remote path (inside the Docker container) is always `/shared/httpd`.

---

**Important:** Remember this, when it comes to path mapping in your IDE/editor configuration.

---

## Configuration

### 1. Install php-debug for Atom

See also:

### 2. Configure path mapping in config.cson or ui

Listing 8: config.cson

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

---

**Important:** On Windows you have to use \\ as directory separators for the local path mapping. E.g.:  
C:\\Users\\projects.

---

## Configure Xdebug for PhpStorm

### Table of Contents

- *Prerequisites*
- *Assumption*
- *Configuration*

### Prerequisites

Ensure that `xdebug.idekey` is set to `PHPSTORM` in your PHP Xdebug configuration.

See also:

- *Configure Xdebug: Docker on Linux*
- *Configure Xdebug: Docker for Mac*
- *Configure Xdebug: Docker for Windows*
- *Configure Xdebug: Docker Toolbox*

## Assumption

For the sake of this example, we will assume the following paths:

Directory	Path
Devilbox git directory	/home/cytopia/repo/devilbox
<i>HOST_PATH_HTTPD_DATADIR</i>	./data/www
Resulting local project path	/home/cytopia/repo/devilbox/data/www

The **Resulting local project path** is the path where all projects are stored locally on your host operating system. No matter what this path is, the equivalent remote path (inside the Docker container) is always `/shared/httpd`.

---

**Important:** Remember this, when it comes to path mapping in your IDE/editor configuration.

---

## Configuration

### 1. Ensure Xdebug port is set to 9000

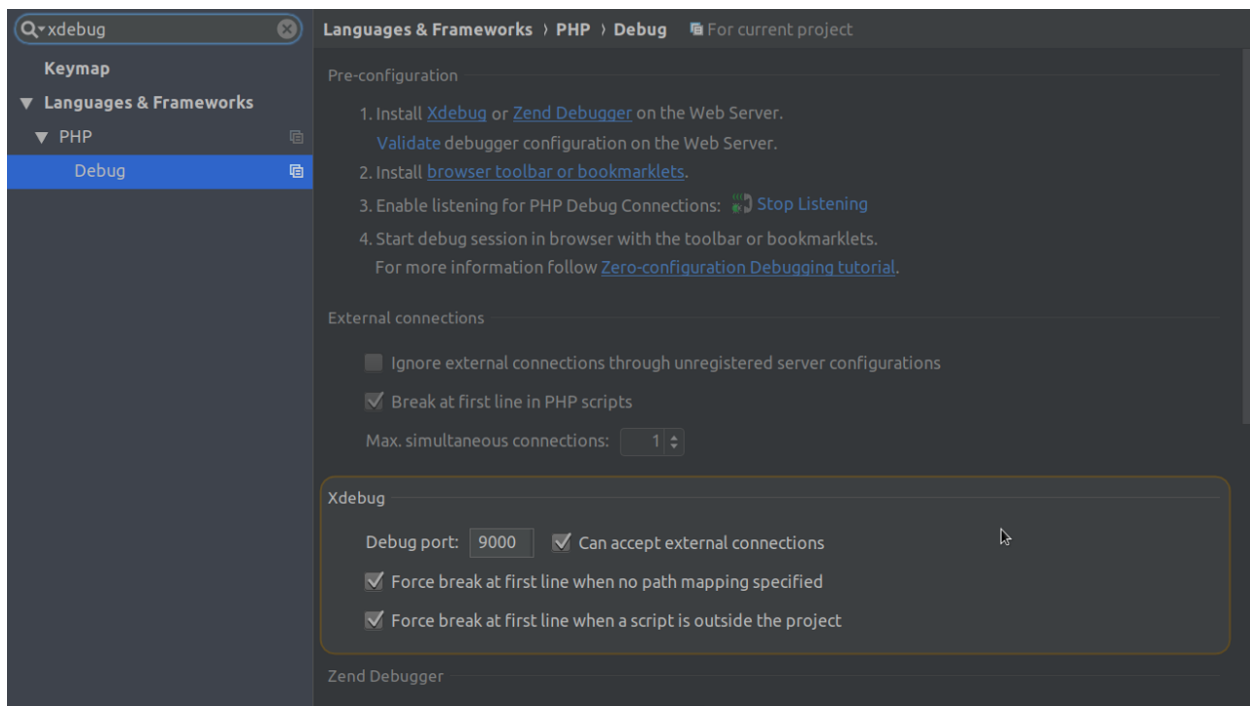


Fig. 1: PhpStorm settings: Xdebug

### 2. Set path mapping

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:

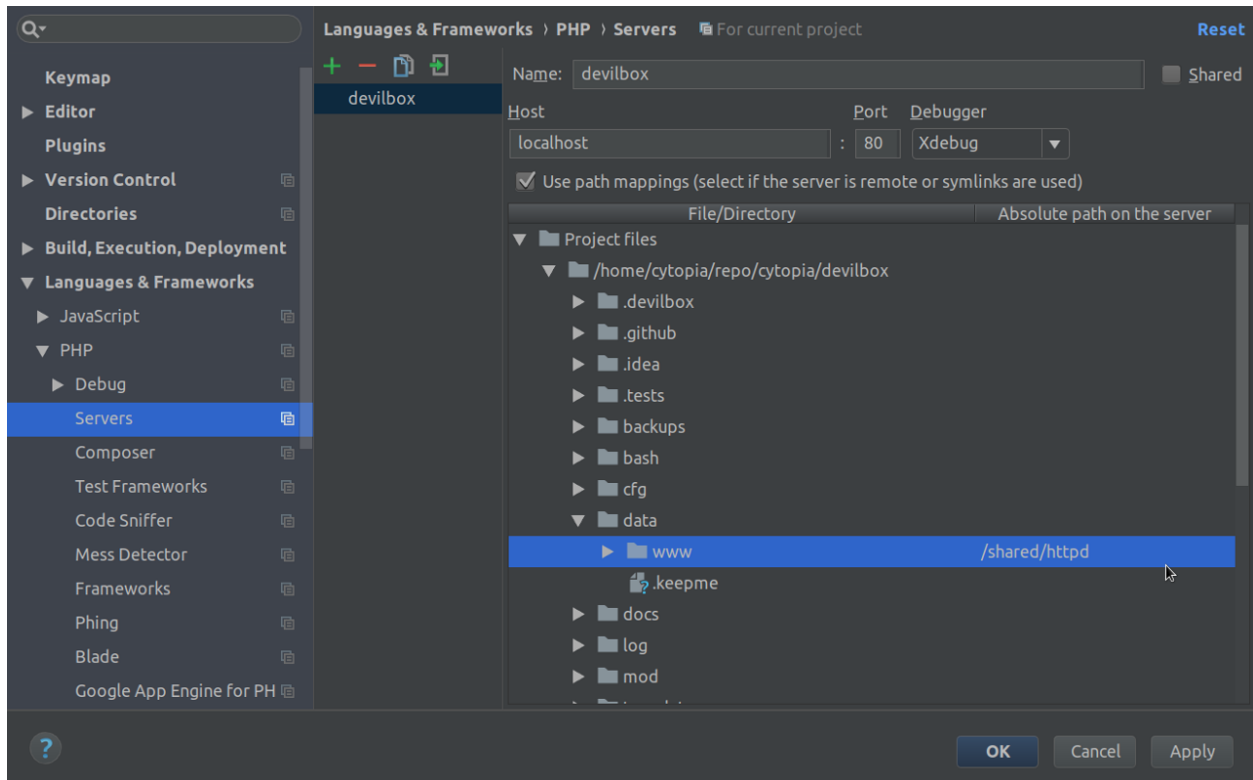


Fig. 2: PhpStorm settings: path mapping

---

**Important:** Recall the path mapping!

---

### 3. Ensure DBGp proxy settings are configured

## Configure Xdebug for Sublime Text 3

### Table of Contents

- *Prerequisites*
- *Assumption*
- *Configuration*

### Prerequisites

Ensure that `xdebug.idekey` is set to `PHPSTORM` in your PHP Xdebug configuration.

See also:

- *Configure Xdebug: Docker on Linux*



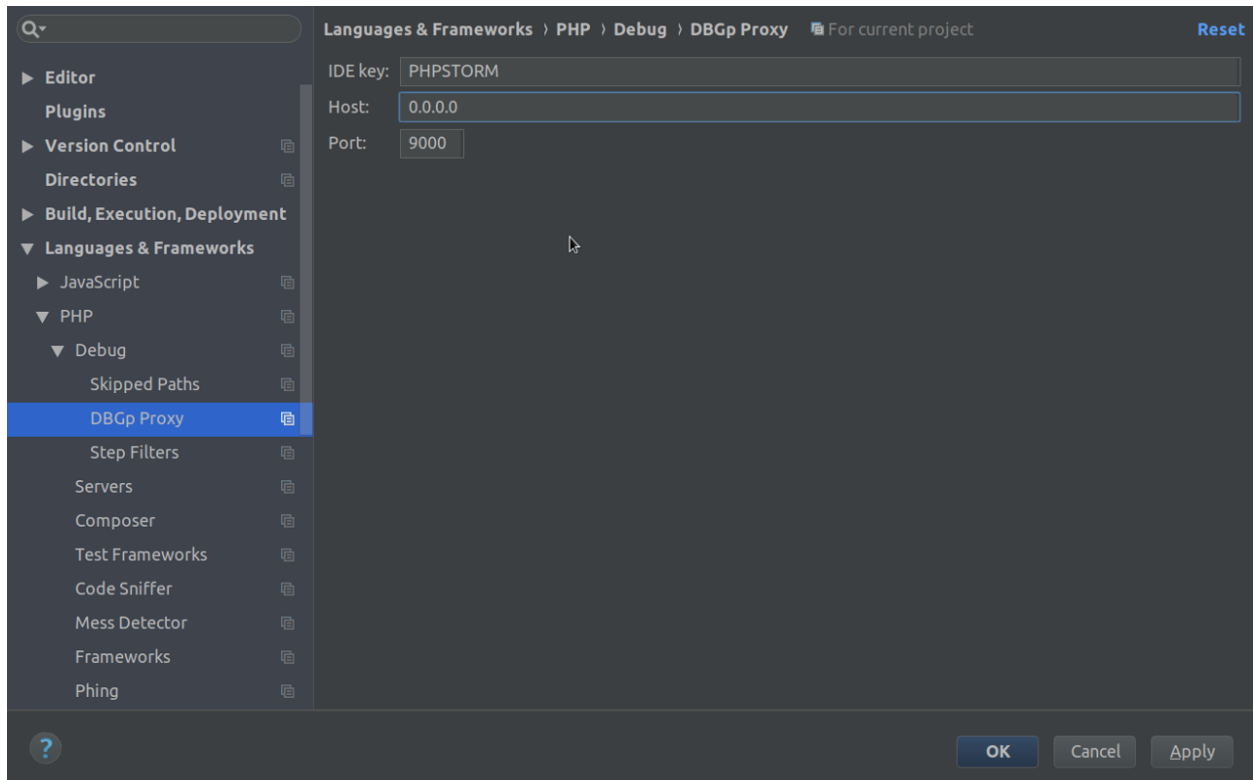


Fig. 3: PHPStorm settings: DBGp Proxy

- *Configure Xdebug: Docker for Mac*
- *Configure Xdebug: Docker for Windows*
- *Configure Xdebug: Docker Toolbox*

## Assumption

For the sake of this example, we will assume the following paths:

Directory	Path
Devilbox git directory	/home/cytopia/repo/devilbox
<i>HOST_PATH_HTTPD_DATADIR</i>	./data/www
Resulting local project path	/home/cytopia/repo/devilbox/data/www

The **Resulting local project path** is the path where all projects are stored locally on your host operating system. No matter what this path is, the equivalent remote path (inside the Docker container) is always /shared/httpd.

---

**Important:** Remember this, when it comes to path mapping in your IDE/editor configuration.

---

## Configuration

### 1. Install Xdebug Client

Use Sublime's Package Control to search for and install Xdebug Client.

**See also:**

## 2. Configure Xdebug.sublime-settings

- Navigate to Tools -> Xdebug -> Settings - User in the menu
- This will open the configuration file in Sublime

Listing 9: Xdebug-sublime-settings

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

---

**Important:** Recall the path mapping!

---

## Configure Xdebug for Visual Studio Code

### Table of Contents

- *Prerequisites*
- *Assumption*
- *Configuration*

### Prerequisites

Ensure that `xdebug.idekey` is set to `PHPSTORM` in your PHP Xdebug configuration.

**See also:**

- *Configure Xdebug: Docker on Linux*
- *Configure Xdebug: Docker for Mac*
- *Configure Xdebug: Docker for Windows*
- *Configure Xdebug: Docker Toolbox*

### Assumption

For the sake of this example, we will assume the following paths:

Directory	Path
Devilbox git directory	/home/cytopia/repo/devilbox
<code>HOST_PATH_HTTPD_DATADIR</code>	./data/www
Resulting local project path	/home/cytopia/repo/devilbox/data/www

The **Resulting local project path** is the path where all projects are stored locally on your host operating system. No matter what this path is, the equivalent remote path (inside the Docker container) is always `/shared/httpd`.

---

**Important:** Remember this, when it comes to path mapping in your IDE/editor configuration.

---

## Configuration

### 1. Install vscode-php-debug

See also:

### 2. Configure launch.json

Listing 10: 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
    }
  ]
}
```

---

**Important:** Recall the path mapping!

---

## 14.3.3 Configuration

See also:

- *Configure Xdebug for Atom*
- *Configure Xdebug for PhpStorm*
- *Configure Xdebug for Sublime Text 3*

- *Configure Xdebug for Visual Studio Code*

## Enable/disable PHP modules

### Table of Contents

- *Enabled PHP modules*
- *Disable PHP modules*
- *Roadmap*

### See also:

<https://github.com/devilbox/docker-php-fpm#user-content-php-modules> Follow the link to see all available PHP modules for each different PHP-FPM server version.

## 15.1 Enabled PHP modules

At the moment all PHP modules are enabled by default except **ioncube**, So this one is the only one you can currently enable. To do so follow the steps provided below:

1. Stop the Devilbox
2. Enable modules in `.env` under `PHP_MODULES_ENABLE`

Listing 1: `.env`

```
# Enable Ioncube
PHP_MODULES_ENABLE=ioncube
```

3. Start the Devilbox

### See also:

*PHP\_MODULES\_ENABLE*

## 15.2 Disable PHP modules

If you feel there are currently too many modules loaded and you want to unload some of them by default, you can do so via a comma separated list in `.env`.

1. Stop the Devilbox
2. Disable modules in `.env` under `PHP_MODULES_DISABLE`

Listing 2: `.env`

```
# Disable Xdebug, Imagick and Swoole
PHP_MODULES_DISABLE=xdebug,imagick,swoole
```

3. Start the Devilbox

**See also:**

*PHP\_MODULES\_DISABLE*

## 15.3 Roadmap

---

**Todo:** In order to create a performant, secure and sane default PHP-FPM server, only really required modules should be enabled by default. The rest is up to the user to enable others as needed.

The current discussion about default modules can be found at the following Github issue. Please participate and give your ideas: <https://github.com/cytopia/devilbox/issues/299>

---

# CHAPTER 16

## 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*

## 16.1 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.

---

## 16.2 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.

---

## 16.3 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



### 17.1 Devilbox Intranet

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.

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.

### 17.2 MailHog

Instead of using the very basic Devilbox intranet UI for emails, you can also enable MailHog and use this to view sent emails.

**See also:**

*Enable and configure MailHog*

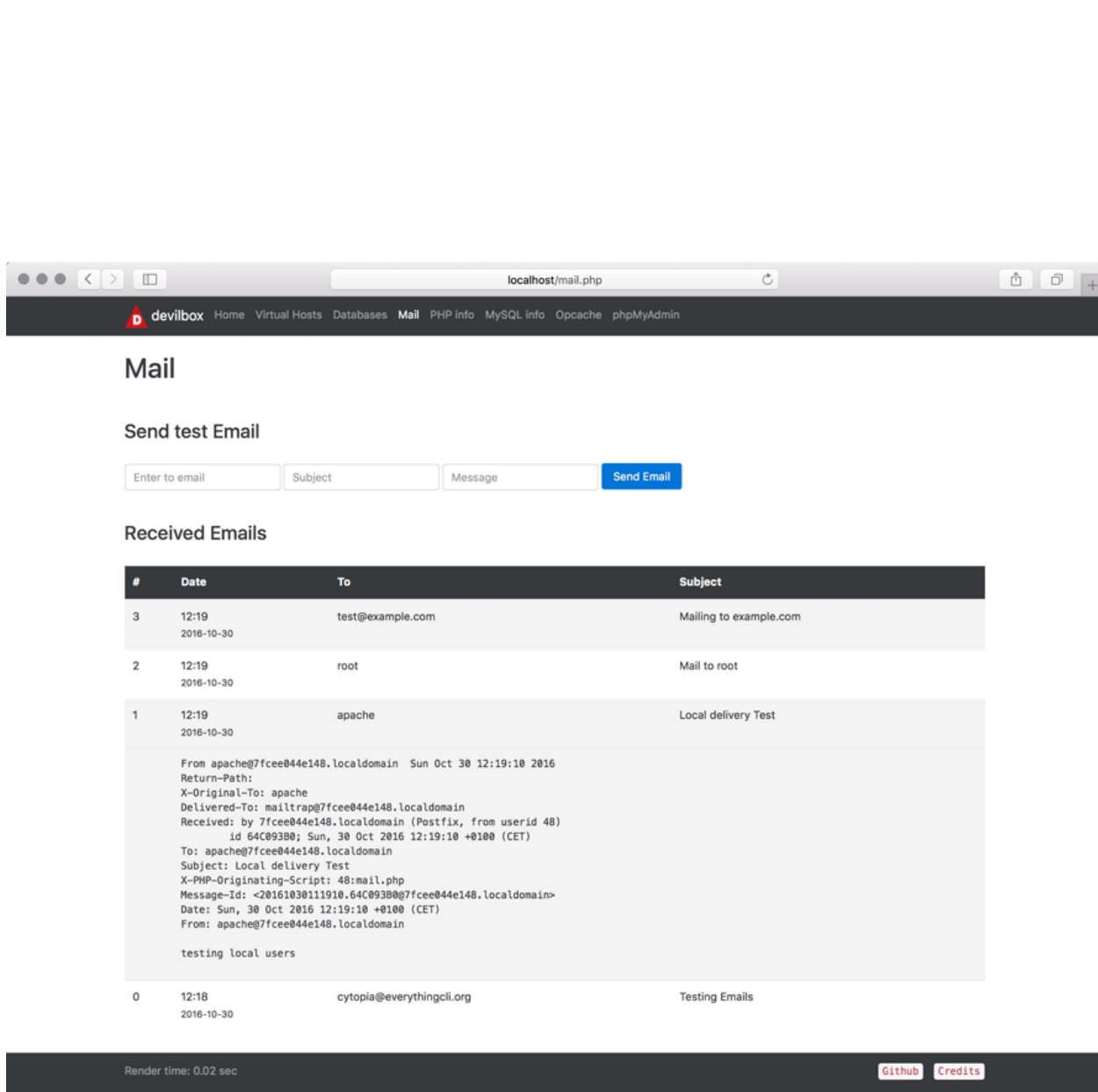


Fig. 1: Devilbox intranet: email catch-all overview

---

## Add 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*

## 18.1 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.

---

## 18.2 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');
?>
```

---

## Work inside the PHP 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*

## 19.1 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`).

### 19.1.1 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 $
```

### 19.1.2 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 $
```

## 19.2 Inside the container

### 19.2.1 `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 synronized container permissions read up here: [Synchronize container permissions](#)

### 19.2.2 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
```

## 19.3 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>
```

## 19.4 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.

### 19.4.1 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`

### 19.4.2 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>

### 19.4.3 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	<code>127.0.0.1</code>	<code>127.0.0.1</code>
Apache/Nginx	<code>127.0.0.1</code>	<code>127.0.0.1</code>
MySQL	<code>127.0.0.1</code>	<code>127.0.0.1</code>
PostgreSQL	<code>127.0.0.1</code>	<code>127.0.0.1</code>
Redis	<code>127.0.0.1</code>	<code>127.0.0.1</code>
Memcached	<code>127.0.0.1</code>	<code>127.0.0.1</code>
MongoDB	<code>127.0.0.1</code>	<code>127.0.0.1</code>

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.

---

**Important:** The mapping of `127.0.0.1` to your host operating system does not work with Docker Toolbox out of the box. In order to achieve the same behaviour read up on: [Docker Toolbox and the Devilbox](#).

---

### 19.4.4 Port mappings

By default, ports are also synronized 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

### 19.4.5 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 [Setup Auto DNS](#).

**See also:**

You can achieve the same on your host operating system by explicitly enabling auto-dns. See also: [Setup Auto DNS](#).

## 19.5 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 synronized
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 and the Devilbox](#)



---

### Source 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 PHP container*

**Table of Contents**

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

## 20.1 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:**

```
# 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
```

## 20.2 PHPCS

PHPCS is a code style analyser for PHP.

**See also:**

```
# 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 .
```

## 20.3 ESLint

ESLint is a Javascript static source code analyzer.

**See also:**

```
# 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 .
```



If you have already operate the Devilbox, this guide is a must have. It will cover common best-practice topics as well as some tips and tricks you will want to apply.

### 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*

## 21.1 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 loosing 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.

### 21.1.1 Projects

See also:

*Move projects to a different directory* Follow this guide to keep your projects separated from the Devilbox git directory.

### 21.1.2 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 1: `.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 2: `.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 3: `.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.

### 21.1.3 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 bein 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.

### 21.1.4 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

## 21.2 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 PHP container*

## 21.3 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.



---

### Customize PHP globally

---

PHP settings can be applied globally to all projects, but are bound to a specific PHP version. This means every PHP version can have its own profile of customized settings.

---

**Note:** By default, all PHP container use roughly the same settings. This might only differ if some options or modules do not exist in a specific container.

---

#### Table of Contents

- *Configure PHP settings globally*
  - *Settings via `php.ini`*
  - *Settings via `php-fpm.conf`*
- *Configure non-overwritable settings globally*
- *Configure loaded PHP modules*
- *Configure PHP-FPM service*

## 22.1 Configure PHP settings globally

PHP settings can either be applied in its `php.ini` configuration file or through the PHP-FPM configuration itself via `php_value` and `php_flag`.

Settings in `php.ini` are also picked up by the PHP command line tool, whereas `php_value` and `php_flag` settings are only valid for requests over the webserver.

This means you can set different values, when executing command line tasks and when the application is run through the browser.

### 22.1.1 Settings via php.ini

To configure PHP globally via php.ini follow the provided link:

**See also:**

*[php.ini](#)*

### 22.1.2 Settings via php-fpm.conf

To configure PHP globally via PHP-FPM follow the provided link:

**See also:**

*[php-fpm.conf](#)*

## 22.2 Configure non-overwritable settings globally

Settings defined via `php.ini`, `php_value` and `php_flag` are applied globally, however they can still be overwritten by any project via the PHP function `ini_set()`.

If you want to create PHP settings and force them, so no application can accidentally or on purpose overwrite them, you need to use `php_admin_value` and `php_admin_flag`.

---

**Important:** Keep in mind that those settings are not picked up by the command line execution of PHP, but only through the browser.

---

To configure PHP globally and non-overwritable via PHP-FPM follow the provided link:

**See also:**

*[php-fpm.conf](#)*

## 22.3 Configure loaded PHP modules

The `.env` file offers the option to specify what PHP modules to enable or disable specifically.

**See also:**

*[Enable/disable PHP modules](#)*

## 22.4 Configure PHP-FPM service

You can also configure the PHP-FPM service itself. Settings can be applied for the core service as well as for the pool. This is useful if you need to adjust performance settings such as number of running child processes, file- and memory limits, timeouts and many more.

Be sure to read up on the PHP-FPM documentation to understand what you are doing.

**See also:**

*[php-fpm.conf](#)*

---

### Customize web server globally

---

Web server settings can be applied globally, which will affect the web server behaviour itself, but not the vhost configuration. Configuration can be done for each version separately, which means each web server can have its own profile of customized settings.

**See also:**

In order to customize the vhosts, have a look at the following links:

- vhost-gen: *Virtual host templates*
- vhost-gen: *Customize all virtual hosts globally*
- vhost-gen: *Customize specific virtual host*
- vhost-gen: *Example: add sub domains*

**Table of Contents**

- *Configure Apache*
- *Configure Nginx*
- *Devilbox specific settings*

### 23.1 Configure Apache

All settings that usually go into the main `httpd.conf` or `apache2.conf` configuration file can be overwritten or customized separately for Apache 2.2 and Apache 2.4.

**See also:**

*apache.conf*

## 23.2 Configure Nginx

All settings that usually go into the main `nginx.conf` configuration file can be overwritten or customized separately for Nginx stable and Nginx mainline.

**See also:**

*nginx.conf*

## 23.3 Devilbox specific settings

There are certain other settings that are directly managed by the Devilbox's `.env` file in order to make other containers aware of those settings.

---

**Important:** Try to avoid to overwrite the `.env` settings via web server configuration files.

---

Use the following `.env` variables to customize this behaviour globally.

**See also:**

- *TLD\_SUFFIX*
- *HOST\_PORT\_HTTPD*
- *HOST\_PORT\_HTTPD\_SSL*
- *HTTPD\_TEMPLATE\_DIR*
- *HTTPD\_DOCROOT\_DIR*



---

### Connect to host OS

---

This section explains how to connect from inside a Devilbox container to the host operating system.

#### Table of Contents

- *Prerequisites*
- *Docker on Linux*
- *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+*
- *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+*
- *Docker Toolbox*
  - *Local port forward on Docker Toolbox*
  - *Remote port-forward on host os*
  - *Post steps*

## 24.1 Prerequisites

When you want to connect from inside a Docker container to a port on your host operating system, ensure the host service is listening on all interfaces for simplicity.

The following sections will give you the IP address and/or the CNAME where the host os can be reached from within a container.

## 24.2 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`).

---

**Important:** Ensure services on the host listen on that IP address or on all interfaces.

---

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`

## 24.3 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:

### 24.3.1 Docker 18.03.0-ce+ and Docker compose 1.20.1+

CNAME: `host.docker.internal`

### 24.3.2 Docker 17.12.0-ce+ and Docker compose 1.18.0+

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

### 24.3.3 Docker 17.06.0-ce+ and Docker compose 1.14.0+

CNAME: `docker.for.mac.localhost`

## 24.4 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:

---

**Important:** Ensure your firewall is not blocking Docker to host connections.

---

### 24.4.1 Docker 18.03.0-ce+ and Docker compose 1.20.1+

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

### 24.4.2 Docker 17.06.0-ce+ and Docker compose 1.14.0+

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

## 24.5 Docker Toolbox

---

**Note:** This section applies for both, Docker Toolbox on MacOS and Docker Toolbox on Windows.

---

Docker Toolbox behaves the same way as Docker on Linux, with one major difference. The Devilbox IP address or the custom provided CNAMEs actually refer to the Docker Toolbox machine.

In order to connect from inside the Docker container (which is inside the Docker Toolbox machine) to your host os, you need to create:

1. either a **local** port-forward on the **Docker Toolbox** machine (`ssh -L`)
2. or a **remote** port-forward on your **host os** (`ssh -R`)

**See also:**

For both examples we assume the following:

- MySQL database exists on your host os and listens on `127.0.0.1` on port `3306`
- Docker Toolbox IP address is `192.168.99.100`
- Host IP address where SSH is listening on `172.16.0.1`
- Host SSH username is `user`
- Devilbox Docker container wants to access MySQL on host os

### 24.5.1 Local port forward on Docker Toolbox

---

**Important:** For that to work, your host operating system requires an SSH server to be up and running.

---

Initiator	From host	From port	To host	To port
Docker Toolbox	<code>127.0.0.1</code>	<code>3306</code>	<code>192.168.99.100</code>	<code>3306</code>

```
# From Docker Toolbox forward port 3306 (on host 172.16.0.1) to myself (192.168.99.
↪100)
toolbox> ssh -L 3306:127.0.0.1:3306 user@172.16.0.1
```

**See also:**

- *Find Docker Toolbox IP address*

- *SSH into Docker Toolbox*
- *SSH port-forward on Docker Toolbox from host*

## 24.5.2 Remote port-forward on host os

---

**Important:** For that to work, your host operating system requires an SSH client (`ssh` binary).

---

Initiator	From host	From port	To host	To port
Host os	127.0.0.1	3306	192.168.99.100	3306

```
# From host os forward port 3306 (from loopback 127.0.0.1) to Docker Toolbox (192.168.
↪99.100)
host> ssh -R 3306:127.0.0.1:3306 docker@192.168.99.100
```

**See also:**

- *Find Docker Toolbox IP address*
- *SSH into Docker Toolbox*
- *SSH port-forward on host to Docker Toolbox*

## 24.5.3 Post steps

With either of the above you have achieved the exact behaviour as *Docker on Linux* for one single service/port (MySQL port 3306).

You must now follow the steps for *Docker on Linux* to actually connect to that service from within the Devilbox Docker container.

---

### Connect to other Docker container

---

Other Docker container can either be accessed by connecting back to the host os or by adding its image directly to the Devilbox stack.

#### Table of Contents

- *Any Docker container on host os*
- *Add Docker container to Devilbox network*
- *Add Docker container to Devilbox stack*

### 25.1 Any Docker container on host os

1. To connect to any other Docker container on your host os from within the Devilbox Docker container, you first need to make sure, you are able to connect to your host os from within the Devilbox Docker container.

#### See also:

*Connect to host OS*

2. Once you are able to connect to the host os, start any other Docker container and make its port that you want to access available to your host os by specifying `-p`. An example with e.g. an external container might look like this:

```
host> docker run -d --name=grafana -p 3000:3000 grafana/grafana
```

You can then connect to your host os on port 3000 from within the Devilbox Docker container and be able to use it.

## 25.2 Add Docker container to Devilbox network

The Devilbox defines its own bridge network, usually called `devilbox_app_net`.

---

**Note:** The name may vary depending on the name of the Devilbox directory. It assembles itself by `<Devilbox_dir_name>_app_net`.

---

1. Start the Devilbox
2. Start your container of choice

```
host> docker run -d --name mycontainer
```

3. Attach your container to the Devilbox network

```
host> docker network connect devilbox_app_net mycontainer
```

Once you have done that, `mycontainer` is then part of the internal Devilbox network and is able to resolve Devilbox container by its name and vice-versa.

4. Connect from Devilbox PHP container to `mycontainer`

From inside the PHP container, you can then refer to your container by its hostname `mycontainer`

## 25.3 Add Docker container to Devilbox stack

Alternatively you can also add any Docker container to the Devilbox network by adding an image it to the Devilbox stack directly.

**See also:**

*Add your own Docker image*

---

### Connect to external hosts

---

Connecting from inside a Devilbox Docker container to any external host works out of the box. The only thing you need is internet/network access and know its hostname or IP address.

Each container has internet access, thus you can `curl`, `fetch`, connect to or download any online resources from within the container.

**See also:**

- *Connect to host OS*
- *Connect to other Docker container*





---

### Add custom CNAME DNS entries

---

You can add an infinite number of custom records that will be available in your running Docker container. If Auto-DNS is turned on, those records will be available on your host operating system as well.

**See also:**

*Setup Auto DNS*

**Table of Contents**

- *Why and what?*
- *How?*

### 27.1 Why and what?

This might be useful if you have an IP address or hostname on your LAN or any other domain which you want to expose to your container by a different CNAME of your choice.

Think of it as setting your `/etc/hosts`, but which will be distributed across all hosts which are using the Devilbox' bundled DNS server.

### 27.2 How?

Adjust the `EXTRA_HOSTS` variable inside `.env` to add as many CNAME's as you need.

As an example, to create a CNAME `mywebserver.com` pointing to `172.16.238.1`, change your `.env` file as shown below:

Listing 1: .env

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

**See also:**

See [EXTRA\\_HOSTS](#) for an in-depth explanation with multiple examples.

---

### 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*

### 28.1 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*

## 28.2 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`)

## 28.3 How to add a new service?

### 28.3.1 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
- 

### 28.3.2 CockroachDB example

Gather the requirements for the Docker image:

1. Name: cockroach
2. Image: cockroachdb/cockroach
3. Tag: latest
4. IP: 172.16.238.240

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: 172.16.238.240
  # For ease of use always automatically start these:
  depends_on:
    - bind
    - php
    - httpd
  # End of custom Docker image
```

## 28.4 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
```

## 28.5 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*

## 29.1 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*

## 29.2 What information do you need?

1. The service to overwrite

## 29.3 How to overwrite a service?

### 29.3.1 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

### 29.3.2 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
```

## 29.4 Further reading

**See also:**

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



---

### Custom scripts per PHP version

---

You can provide custom startup commands via bash scripts to each of the PHP container individually. This may be useful to specify additional software to install or additional settings to apply during the initial startup.

**See also:**

- *Custom scripts globally* (equal for all PHP versions)
- *Autostarting NodeJS Apps*

---

**Note:** Per PHP version scripts are always executed **before** global scripts.

---

#### Table of Contents

- *General*
  - *Where*
  - *When*
  - *How*
- *Examples*
  - *Installing Microsoft ODBC driver*
  - *Running commands as devilbox user*

### 30.1 General

You can add custom shell scripts for each PHP version separately.

---

**Important:** Provided scripts must end by the file extension `.sh` and should be executable. Anything not ending by `.sh` will be ignored.

---

---

**Important:** Provided scripts will be executed by the `root` user within the PHP container.

---

### 30.1.1 Where

Startup scripts can be added to `cfg/php-startup-X.Y/`. See the directory structure for PHP startup script directories inside `./cfg/` directory:

```
host> ls -l path/to/devilbox/cfg/ | grep 'php-startup'
```

drwxr-xr-x	2	cytopia	cytopia	4096	Mar	5	21:53	php-startup-5.2/
drwxr-xr-x	2	cytopia	cytopia	4096	Mar	5	21:53	php-startup-5.3/
drwxr-xr-x	2	cytopia	cytopia	4096	Mar	5	21:53	php-startup-5.4/
drwxr-xr-x	2	cytopia	cytopia	4096	Mar	5	21:53	php-startup-5.5/
drwxr-xr-x	2	cytopia	cytopia	4096	Apr	3	22:04	php-startup-5.6/
drwxr-xr-x	2	cytopia	cytopia	4096	Mar	5	21:53	php-startup-7.0/
drwxr-xr-x	2	cytopia	cytopia	4096	Mar	5	21:53	php-startup-7.1/
drwxr-xr-x	2	cytopia	cytopia	4096	Mar	5	21:53	php-startup-7.2/
drwxr-xr-x	2	cytopia	cytopia	4096	Mar	5	21:53	php-startup-7.3/
drwxr-xr-x	2	cytopia	cytopia	4096	Mar	5	21:53	php-startup-7.4/

Custom scripts are added by placing a file into `cfg/php-startup-X.X/` (where `X.X` stands for your PHP version). The file must end by `.sh` in order to be executed by the PHP container.

Some of the PHP startup directories contain a few example files with the file suffix `-example`. If you want to use them, copy these files to a new name without the `-example` suffix and ensure they end by `.sh`.

### 30.1.2 When

The scripts will be executed by the PHP container during initial startup. Whenever you change your scripts, ensure to restart the Devilbox.

### 30.1.3 How

The scripts will always be executed inside the PHP container (Debian Linux) and will be run with `root` privileges. It is however possible to drop privileges within the script to have them executed as a normal user.

## 30.2 Examples

### 30.2.1 Installing Microsoft ODBC driver

This example will add Microsofts ODBC driver to PHP 7.1. These drivers are required in order to make the PHP modules `pdo_sqlsrv` and `sqlsrv` work. The two mentioned modules are already available in the PHP container, but are explicitly disabled via `PHP_MODULES_DISABLE`.

They won't work without the ODBC driver installed, which unfortunately cannot be bundled, as it requires every user to accept a license/EULA by Microsoft.

```
# Navigate to startup dir of PHP 7.1
host> cd path/to/devilbox/cfg/php-startup-7.1

# Create an .sh file
host> touch ms-odbc.sh

# Open the file in your favourite editor
host> vi ms-odbc.sh
```

Paste the following into `ms-odbc.sh` and **ensure to accept the EULA** by changing `ACCEPT_EULA=N` to `ACCEPT_EULA=Y`.

Listing 1: `cfg/php-startup-7.1/install-ms-odbc.sh`

```
#!/bin/bash
#
# This script will automatically install the Microsoft ODBC driver for MsSQL
# support for PHP during startup.
#
# In order for it to work, you must read and accept their License/EULA:
# https://odbcsql.blob.core.windows.net/eula17/LICENSE172.TXT
#

# -----
# EDIT THE VARIABLE BELOW TO ACCEPT THE EULA (If you agree to their terms)
# -----
#

###
### Set this to "Y" (capital 'Y') if you accept the EULA.
###
ACCEPT_EULA=N

# -----
# DO NOT EDIT BELOW THIS LINE
# -----
#

###
### Where to retrieve the deb package
###
MSODBC_URL="https://packages.microsoft.com/debian/8/prod/pool/main/m/msodbcsql17/"

###
### Pre-flight check
###
if [ "${ACCEPT_EULA}" != "Y" ]; then
    echo "MS ODBC EULA not accepted. Aborting installation."
    exit 0
```

(continues on next page)

(continued from previous page)

```
fi

###
### EULA accepted, so we can proceed
###

# Extract latest *.deb packate
MSODBC_DEB="$( curl -k -sS "${MSODBC_URL}" | grep -Eo 'msodbcsql[-._0-9]+?_amd64\.deb
↪' | tail -1 )"

# Download to temporary location
curl -k -sS "${MSODBC_URL}/${MSODBC_DEB}" > "/tmp/${MSODBC_DEB}"

# Install
ACCEPT_EULA="${ACCEPT_EULA}" dpkg -i "/tmp/${MSODBC_DEB}"

# Remove artifacts
rm -f "/tmp/${MSODBC_DEB}"
```

---

**Important:** The script will not work, if you have not accepted the EULA.

---

### 30.2.2 Running commands as devilbox user

As mentioned above, all scripts are run by the `root` user. If you do need something to be executed as the normal user: `devilbox`, you can simply `su` inside the shell script.

The following example will install `grunt` and start a NodeJS application as the `devilbox` user for the PHP 7.1 Docker container only.

Listing 2: `cfg/php-startup-7.1/myscript.sh`

```
# Install grunt as devilbox user
su -c "npm install grunt" -l devilbox

# Start a NodeJS application with pm2 as devilbox user
su -c "cd /shared/httpd/my-node/src/; pm2 start index.js" -l devilbox
```

---

## Custom scripts globally

---

You can provide custom startup commands via bash scripts that are executed by all PHP container. This may be useful to specify additional software to install or additional settings to apply during the initial startup.

**See also:**

- *Custom scripts per PHP version* (individually for different PHP versions)
- *Autostarting NodeJS Apps*

---

**Note:** Global scripts are always executed **after** per PHP version scripts.

---

### Table of Contents

- *General*
  - *Where*
  - *When*
  - *How*
- *Examples*
  - *Running commands as devilbox user*

## 31.1 General

You can add shell scripts that are executed for all PHP container equally.

---

**Important:** Provided scripts must end by the file extension `.sh` and should be executable. Anything not ending by

---

.sh will be ignored.

---

**Important:** Provided scripts will be executed by the `root` user within the PHP container.

---

### 31.1.1 Where

Startup scripts can be added to `autostart/`.

```
host> ls -l path/to/devilbox/

drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 autostart/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 backups/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 bash/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 ca/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 cfg/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 compose/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 data/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 docs/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 mail/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 mod/
```

Custom scripts are added by placing a file into `autostart/`. The file must end by `.sh` in order to be executed by the PHP container.

### 31.1.2 When

The scripts will be executed by the PHP container during initial startup. Whenever you change your scripts, ensure to restart the Devilbox.

### 31.1.3 How

The scripts will always be executed inside the PHP container (Debian Linux) and will be run with `root` privileges. It is however possible to drop privileges within the script to have them executed as a normal user.

## 31.2 Examples

### 31.2.1 Running commands as devilbox user

As mentioned above, all scripts are run by the `root` user. If you do need something to be executed as the normal user: `devilbox`, you can simply `su` inside the shell script.

The following example will install `grunt` and start a NodeJS application as the `devilbox` user for whatever PHP container has been started.



Listing 1: autostart/myscript.sh

```
# Install grunt as devilbox user
su -c "npm install grunt" -l devilbox

# Start a NodeJS application with pm2 as devilbox user
su -c "cd /shared/httpd/my-node/src/; pm2 start index.js" -l devilbox
```



---

### Autostarting NodeJS Apps

---

You can have all of your NodeJS applications spin up automatically as soon as you `docker-compose up`. This can be achieved by making use of (Node.js Process Manager) and the autostart feature.

**See also:**

**Read more about how to add scripts for autostart commands:**

- *Custom scripts per PHP version* (individually for different PHP versions)
- *Custom scripts globally* (equal for all PHP versions)

**Table of Contents**

- *Self-built*
  - *Assumption*
  - *The script*
- *Pre-built*
- *Reverse proxy NodeJS*

## 32.1 Self-built

Simply add a script ending by `.sh` to the `autostart/` directory that will accomplish this. The following example will make use of to spin up your NodeJS application.

### 32.1.1 Assumption

- Path to your NodeJS project (within the Docker container): `/shared/httpd/my-node/src`
- Name of the JS file to startup: `index.js`

### 32.1.2 The script

Add the following script to `autostart/`

Listing 1: `autostart/myscript.sh`

```
su -c "cd /shared/httpd/my-node/src; pm2 start index.js" -l devilbox
```

- The whole command is wrapped into `su` to ensure the application will be started as the user `devilbox`.
- `cd` tells it to you enter the directory where `index.js` can be found
- And finally will take care about starting up your javascript file.

Once the Devilbox is running, you can enter the PHP container and verify with `pm2 list` that everything is running as expected.

## 32.2 Pre-built

Instead of writing multiple scripts for multiple applications, you can also make use of the pre-shipped script that allows you to start unlimited NodeJS applications via .

The following script is provided in `autostart/run-node-js-projects.sh-example` and needs to be copied to a file ending by `.sh`

```
host> cd /path/to/devilbox
host> cd autostart
host> cp run-node-js-projects.sh-example run-node-js-projects.sh
```

In that newly created file, you can simply add the full paths (path inside the Docker containre) of your Javascript files that need to be started. There is already one example which is not commented. Change this to your path and add as many lines as you have projects to startup.

Listing 2: `autostart/run-node-js-projects.sh`

```
#!/usr/bin/env bash
#
# This is a generic example to startup your NodeJS projects with
# pm2 (https://github.com/Unitech/pm2)
#
# Important: As everything is run by the root user, you must explicitly direct the
#            commands to the devilbox user.
#
#
# Add the full paths of your Nodejs projects startup files into this array
# Each project separated by a newline and enclosed in double quotes. (No commas!)
# Paths are internal paths inside the PHP container.
NODE_PROJECTS=(
    #"/shared/httpd/my-rhost/js/index.js"
    #"/shared/httpd/my-node-hello-world/name/run.js"
    #"/shared/httpd/another-node-project/javascript/run.js"
)

# Check if any projects have been defined
```

(continues on next page)

(continued from previous page)

```

if [ ${#NODE_PROJECTS[@]} -eq 0 ]; then
    echo "No projects defined. Exiting."
    exit 0
fi

# This loops over the paths, separates base directory and filename and will run it in_
↳the background
# as the user devilbox. There shouldn't be any need to change anything here.
for item in ${NODE_PROJECTS[*]}; do
    NODE_PATH="$( dirname "${item}" )"
    NODE_FILE="$( basename "${item}" )"

    if [ ! -d "${NODE_PATH}" ]; then
        >&2 echo "[Warning], skipping startup, directory does not exist: $
↳{NODE_PATH}"
        continue;
    fi
    if [ ! -f "${NODE_PATH}/${NODE_FILE}" ]; then
        >&2 echo "[Warning], skipping startup, file does not exist: ${NODE_
↳PATH}/${NODE_FILE}"
        continue;
    fi

    echo "su -c \"cd ${NODE_PATH}; pm2 start ${NODE_FILE}\" -l devilbox"
    su -c "cd ${NODE_PATH}; pm2 start ${NODE_FILE}" -l devilbox
done

```

## 32.3 Reverse proxy NodeJS

If you also want to know how to reverse proxy your NodeJS service and have it available via the web server including HTTPS support have a look at the following links:

### See also:

- [Reverse Proxy with HTTPS](#)
- [Setup reverse proxy NodeJS](#)

Imagine you have started an application within the PHP container that creates a listening port (e.g.: NodeJS). This will now only listen on the PHP container and you would have to adjust the docker-compose.yml definition in order to have that port available outside to your host OS.

Alternatively, there is a simple way to reverse proxy it to the already running web server and even make use of the already available HTTPS feature.



---

### Virtual host templates

---

#### Table of Contents

- *Overview*
  - *What is it?*
  - *Template files*
    - \* *Normal virtual host*
    - \* *Reverse proxy*
  - *Template sections*
- *Virtual host Templates*
  - *Apache 2.2 template*
  - *Apache 2.4 template*
  - *Nginx template*
- *Reverse proxy Templates*
  - *Apache 2.2 template*
  - *Apache 2.4 template*
  - *Nginx template*

## 33.1 Overview

### 33.1.1 What is it?

vhost-gen templates are yaml files which contain a general definition for a virtual host definition. Those templates contain placeholders in the form of `__<NAME>__` which will be replaced by settings applied to the Devilbox.

**See also:**

### 33.1.2 Template files

By default, vhost-gen templates are located within the Devilbox root directory under `cfg/vhost-gen/`. The templates file names are suffixed with `-example-<type>` and are absolutely identical to what is shipped inside each Devilbox web server Docker container.

---

**Note:** Also note that nginx stable and nginx mainline share the same template as their configuration syntax is identical.

---

#### Normal virtual host

All template files ending by `-example-vhost` can be used to customize a normal file serving virtual host.

```
host> tree -L 1 cfg/vhost-gen/

cfg/vhost-gen/
├── apache22.yml-example-rproxy
├── apache22.yml-example-vhost
├── apache24.yml-example-rproxy
├── apache24.yml-example-vhost
├── nginx.yml-example-rproxy
├── nginx.yml-example-vhost
└── README.md

0 directories, 7 files
```

#### Reverse proxy

All template files ending by `-example-rproxy` can be used to create a reverse proxy for your project.

```
host> tree -L 1 cfg/vhost-gen/

cfg/vhost-gen/
├── apache22.yml-example-rproxy
├── apache22.yml-example-vhost
├── apache24.yml-example-rproxy
├── apache24.yml-example-vhost
├── nginx.yml-example-rproxy
├── nginx.yml-example-vhost
└── README.md

0 directories, 7 files
```



### 33.1.3 Template sections

All vhost-gen templates consist of three sections:

Section	Description
vhost	This is the part that is actually rendered into the vhost configuration. All other sections will be inserted into this one.
vhost_type	The vhost type determines the type of vhost: reverse proxy or document root based vhost. The Devilbox currently does not support reverse proxy vhost.
features	The feature section contains many sub-sections that are replaced into the vhost section before final rendering.

## 33.2 Virtual host Templates

These templates can be used to alter the behaviour of the vhost on a per project base or globally.

### 33.2.1 Apache 2.2 template

Listing 1: apache22.yml-example-vhost

```

---
# Apache 2.2 vHost Template defintion for vhost-gen.py
#
# The 'feature' section contains optional features that can be enabled via
# conf.yml and will then be replaced into the main vhost ('structure' section)
# into their corresponding position:
#
#   __XDOMAIN_REQ__
#   __PHP_FPM__
#   __ALIASES__
#   __DENIES__
#   __STATUS__
#
# The features itself also contain variables to be adjusted in conf.yml
# and will then be replaced in their corresponding feature section
# before being replaced into the vhost section (if enabled):
#
# PHP-FPM:
#   __PHP_ADDR__
#   __PHP_PORT__
# XDomain:
#   __REGEX__
# Alias:
#   __REGEX__
#   __PATH__
# Deny:
#   __REGEX__
# Status:
#   __REGEX__
#
# Variables to be replaced directly in the vhost configuration can also be set
# in conf.yml and include:

```

(continues on next page)

(continued from previous page)

```

#   __VHOST_NAME__
#   __DOCUMENT_ROOT__
#   __INDEX__
#   __ACCESS_LOG__
#   __ERROR_LOG__
#   __PHP_ADDR__
#   __PHP_PORT__
#

###
### Basic vHost skeleton
###
vhost: |
  <VirtualHost __DEFAULT_VHOST__:__PORT__>
    ServerName   __VHOST_NAME__

    CustomLog    "__ACCESS_LOG__" combined
    ErrorLog     "__ERROR_LOG__"

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

###
### vHost Type (normal or reverse proxy)
###
vhost_type:
  # Normal vHost (-p)
  docroot: |
    # Define the vhost to serve files
    DocumentRoot "__DOCUMENT_ROOT__"
    <Directory "__DOCUMENT_ROOT__">
      DirectoryIndex __INDEX__

      AllowOverride All
      Options All

      RewriteEngine on
      RewriteBase /

      Order allow,deny
      Allow from all
    </Directory>

  # Reverse Proxy (-r)
  rproxy: |
    # Define the vhost to reverse proxy
    ProxyRequests On

```

(continues on next page)

(continued from previous page)

```

ProxyPreserveHost On
ProxyPass __LOCATION__ __PROXY_PROTO__://__PROXY_ADDR__:__PROXY_PORT__LOCATION__
ProxyPassReverse __LOCATION__ __PROXY_PROTO__://__PROXY_ADDR__:__PROXY_PORT__
↪LOCATION__

###
### Optional features to be enabled in vHost
###
features:

# SSL Configuration
ssl: |
    SSLEngine on
    SSLCertificateFile    "__SSL_PATH_CRT__"
    SSLCertificateKeyFile "__SSL_PATH_KEY__"
    SSLProtocol           __SSL_PROTOCOLS__
    SSLHonorCipherOrder   __SSL_HONOR_CIPHER_ORDER__
    SSLCipherSuite        __SSL_CIPHERS__

# Redirect to SSL directive
redirect: |
    RedirectMatch (.*) https://__VHOST_NAME__:__SSL_PORT__$1

# PHP-FPM will not be applied to a reverse proxy!
php_fpm: |
    # PHP-FPM Definition
    ProxyPassMatch ^/(.*\.php(/.*)?)$ fcgi://__PHP_ADDR__:__PHP_PORT__DOCUMENT_ROOT_
↪_/$1 timeout=__PHP_TIMEOUT__

alias: |
    # Alias Definition
    Alias "__ALIAS__" "__PATH__ALIAS__"
    <Location "__ALIAS__">
        __XDOMAIN_REQ__
    </Location>
    <Directory "__PATH__ALIAS__">
        Order allow,deny
        Allow from all
    </Directory>

deny: |
    # Deny Definition
    <FilesMatch "__REGEX__">
        Order allow,deny
        Deny from all
    </FilesMatch>

server_status: |
    # Status Page
    <Location __REGEX__>
        SetHandler server-status
        Order allow,deny
        Allow from all
    </Location>

xdomain_request: |

```

(continues on next page)

(continued from previous page)

```

# Allow cross domain request from these hosts
SetEnvIf Origin "__REGEX__" AccessControlAllowOrigin=$0
Header add Access-Control-Allow-Origin %{AccessControlAllowOrigin}e
↪env=AccessControlAllowOrigin
Header always set Access-Control-Allow-Methods "POST, GET, OPTIONS, DELETE, PUT"
Header always set Access-Control-Max-Age "0"
Header always set Access-Control-Allow-Headers "x-requested-with, Content-Type,
↪origin, authorization, accept, client-security-token"
# Added a rewrite to respond with a 200 SUCCESS on every OPTIONS request.
RewriteEngine On
RewriteCond %{REQUEST_METHOD} OPTIONS
RewriteRule ^(.*)$ $1 [R=200,L]

```

### 33.2.2 Apache 2.4 template

Listing 2: apache24.yml-example-vhost

```

---
# Apache 2.4 vHost Template defintion for vhost-gen.py
#
# The 'feature' section contains optional features that can be enabled via
# conf.yml and will then be replaced into the main vhost ('structure' section)
# into their corresponding position:
#
#   __XDOMAIN_REQ__
#   __PHP_FPM__
#   __ALIASES__
#   __DENIES__
#   __STATUS__
#
# The features itself also contain variables to be adjusted in conf.yml
# and will then be replaced in their corresponding feature section
# before being replaced into the vhost section (if enabled):
#
# PHP-FPM:
#   __PHP_ADDR__
#   __PHP_PORT__
# XDomain:
#   __REGEX__
# Alias:
#   __REGEX__
#   __PATH__
# Deny:
#   __REGEX__
# Status:
#   __REGEX__
#
# Variables to be replaced directly in the vhost configuration can also be set
# in conf.yml and include:
#   __VHOST_NAME__
#   __DOCUMENT_ROOT__
#   __INDEX__
#   __ACCESS_LOG__
#   __ERROR_LOG__

```

(continues on next page)

(continued from previous page)

```

#   __PHP_ADDR__
#   __PHP_PORT__
#

###
### Basic vHost skeleton
###
vhost: |
  <VirtualHost __DEFAULT_VHOST__:__PORT__>
    ServerName __VHOST_NAME__
    Protocols __HTTP_PROTO__

    CustomLog "__ACCESS_LOG__" combined
    ErrorLog  "__ERROR_LOG__"

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

###
### vHost Type (normal or reverse proxy)
###
vhost_type:
  # Normal vHost (-p)
  docroot: |
    # Define the vhost to serve files
    DocumentRoot "__DOCUMENT_ROOT__"
    <Directory "__DOCUMENT_ROOT__">
      DirectoryIndex __INDEX__

      AllowOverride All
      Options All

      RewriteEngine on
      RewriteBase /

      Order allow,deny
      Allow from all
      Require all granted
    </Directory>

  # Reverse Proxy (-r)
  rproxy: |
    # Define the vhost to reverse proxy
    ProxyRequests On
    ProxyPreserveHost On
    ProxyPass __LOCATION__ __PROXY_PROTO__:__PROXY_ADDR__:__PROXY_PORT____LOCATION__
    ProxyPassReverse __LOCATION__ __PROXY_PROTO__:__PROXY_ADDR__:__PROXY_PORT__
    ↪__LOCATION__

```

(continues on next page)

(continued from previous page)

```

###
### Optional features to be enabled in vHost
###
features:

# SSL Configuration
ssl: |
    SSLEngine on
    SSLCertificateFile    "__SSL_PATH_CRT__"
    SSLCertificateKeyFile "__SSL_PATH_KEY__"
    SSLProtocol           __SSL_PROTOCOLS__
    SSLHonorCipherOrder   __SSL_HONOR_CIPHER_ORDER__
    SSLCipherSuite        __SSL_CIPHERS__

# Redirect to SSL directive
redirect: |
    RedirectMatch (.* ) https://__VHOST_NAME__:__SSL_PORT__$1

# PHP-FPM will not be applied to a reverse proxy!
php_fpm: |
    # In case for PHP-FPM 5.2 compatibility use 'GENERIC' instead of 'FPM'
    # https://httpd.apache.org/docs/2.4/mod/mod_proxy_fcgi.html#proxyfcgibackendtype
    ProxyFCGIBackendType FPM

    # PHP-FPM Definition
    <FilesMatch \.php$>
        Require all granted
        SetHandler proxy:fcgi://__PHP_ADDR__:__PHP_PORT__
    </FilesMatch>

    <Proxy "fcgi://__PHP_ADDR__:__PHP_PORT__/">
        ProxySet timeout=__PHP_TIMEOUT__
        ProxySet connectiontimeout=__PHP_TIMEOUT__
    </Proxy>

    # If the php file doesn't exist, disable the proxy handler.
    # This will allow .htaccess rewrite rules to work and
    # the client will see the default 404 page of Apache
    RewriteCond %{REQUEST_FILENAME} \.php$
    RewriteCond %{DOCUMENT_ROOT}/%{REQUEST_URI} !-f
    RewriteRule (.* ) - [H=text/html]

alias: |
    # Alias Definition
    Alias "__ALIAS__" "__PATH__ALIAS__"
    <Location "__ALIAS__">
        __XDOMAIN_REQ__
    </Location>
    <Directory "__PATH__ALIAS__">
        Order allow,deny
        Allow from all
        Require all granted
    </Directory>

deny: |

```

(continues on next page)

(continued from previous page)

```

# Deny Definition
<FilesMatch "__REGEX__">
    Order allow,deny
    Deny from all
</FilesMatch>

server_status: |
# Status Page
<Location __REGEX__>
    SetHandler server-status
    Order allow,deny
    Allow from all
    Require all granted
</Location>

xdomain_request: |
# Allow cross domain request from these hosts
SetEnvIf Origin "__REGEX__" AccessControlAllowOrigin=$0
Header add Access-Control-Allow-Origin %{AccessControlAllowOrigin}e
↪env=AccessControlAllowOrigin
Header always set Access-Control-Allow-Methods "POST, GET, OPTIONS, DELETE, PUT"
Header always set Access-Control-Max-Age "0"
Header always set Access-Control-Allow-Headers "x-requested-with, Content-Type,
↪origin, authorization, accept, client-security-token"
# Added a rewrite to respond with a 200 SUCCESS on every OPTIONS request.
RewriteEngine On
RewriteCond %{REQUEST_METHOD} OPTIONS
RewriteRule ^(.*)$ $1 [R=200,L]

```

### 33.2.3 Nginx template

Listing 3: nginx.yml-example-vhost

```

---
# Nginx vHost Template defintion for vhost-gen.py
#
# The 'feature' section contains optional features that can be enabled via
# conf.yml and will then be replaced into the main vhost ('structure' section)
# into their corresponding position:
#
#   __XDOMAIN_REQ__
#   __PHP_FPM__
#   __ALIASES__
#   __DENIES__
#   __STATUS__
#
# The features itself also contain variables to be adjusted in conf.yml
# and will then be replaced in their corresponding feature section
# before being replaced into the vhost section (if enabled):
#
# PHP-FPM:
#   __PHP_ADDR__
#   __PHP_PORT__
# XDomain:

```

(continues on next page)

(continued from previous page)

```

#   __REGEX__
# Alias:
#   __REGEX__
#   __PATH__
# Deny:
#   __REGEX__
# Status:
#   __REGEX__
#
# Variables to be replaced directly in the vhost configuration can also be set
# in conf.yml and include:
#   __VHOST_NAME__
#   __DOCUMENT_ROOT__
#   __INDEX__
#   __ACCESS_LOG__
#   __ERROR_LOG__
#   __PHP_ADDR__
#   __PHP_PORT__
#
###
### Basic vHost skeleton
###
vhost: |
  server {
    listen      __PORT__ __HTTP_PROTO__ __DEFAULT_VHOST__;
    server_name __VHOST_NAME__;

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

    __REDIRECT__
    __SSL__
    __VHOST_DOCROOT__
    __VHOST_RPROXY__
    __PHP_FPM__
    __ALIASES__
    __DENIES__
    __SERVER_STATUS__
    # Custom directives
    __CUSTOM__
  }

###
### vHost Type (normal or reverse proxy)
###
vhost_type:
  # Normal vHost (-p)
  docroot: |
    # Define the vhost to serve files
    root      "__DOCUMENT_ROOT__";
    index      __INDEX__;

  # Reverse Proxy (-r)
  rproxy: |

```

(continues on next page)



(continued from previous page)

```

# Define the vhost to reverse proxy
location __LOCATION__ {
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_pass __PROXY_PROTO__://__PROXY_ADDR__:__PROXY_PORT__;
}

###
### Optional features to be enabled in vHost
###
features:

# SSL Configuration
ssl: |
    ssl_certificate      __SSL_PATH_CRT__;
    ssl_certificate_key  __SSL_PATH_KEY__;
    ssl_protocols        __SSL_PROTOCOLS__;
    ssl_prefer_server_ciphers __SSL_HONOR_CIPHER_ORDER__;
    ssl_ciphers          __SSL_CIPHERS__;

# Redirect to SSL directive
redirect: |
    return 301 https://__VHOST_NAME__:__SSL_PORT__$request_uri;

# PHP-FPM will not be applied to a reverse proxy!
php_fpm: |
    # PHP-FPM Definition
    location / {
        try_files $uri $uri/ /index.php$?is_args$args;
    }
    location ~ /\.php?$ {
        try_files $uri = 404;
        include fastcgi_params;

        # https://stackoverflow.com/questions/1733306/nginx-errors-readv-and-recv-
        ↪failed/51457613#51457613
        fastcgi_keep_conn off;

        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_split_path_info ^(.+\.php)(.*)$;

        fastcgi_pass __PHP_ADDR__:__PHP_PORT__;
        fastcgi_read_timeout __PHP_TIMEOUT__;

        fastcgi_index index.php;
        fastcgi_intercept_errors on;
    }

alias: |
    # Alias Definition
    location ~ __ALIAS__ {
        root __PATH__;
        __XDOMAIN_REQ__
    }

deny: |

```

(continues on next page)

(continued from previous page)

```
# Deny Definition
location ~ __REGEX__ {
    deny all;
}

server_status: |
# Status Page
location ~ __REGEX__ {
    stub_status on;
    access_log off;
}

xdomain_request: |
# Allow cross domain request from these hosts
if ( $http_origin ~* (__REGEX__) ) {
    add_header "Access-Control-Allow-Origin" "$http_origin";
    add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
    add_header 'Access-Control-Allow-Headers' 'DNT,X-CustomHeader,Keep-Alive,User-
↪Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Content-
↪Range';
    add_header 'Access-Control-Expose-Headers' 'DNT,X-CustomHeader,Keep-Alive,
↪User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Content-
↪Range,Range';
    add_header 'Access-Control-Max-Age' 0;
    return 200;
}
```

## 33.3 Reverse proxy Templates

These templates can be used to change a normal vhost into a reverse proxy project. This might be useful if you use NodeJs applications for example.

---

**Important:** Do not apply those templates globally. They are intended to be used on a per project base.

---

---

**Note:** In order to use the Reverse Proxy templates you will only need to adjust the listening port, everything else will work as already defined. So you simply need to copy those files into your project directory. Lines that need to be changed are marked below. The currently set default listening port is 8000.

---

### 33.3.1 Apache 2.2 template

Listing 4: apache22.yml-example-rproxy

```
---
# Apache 2.2 Reverse Proxy Template defintion for vhost-gen.py
#
# The 'feature' section contains optional features that can be enabled via
# conf.yml and will then be replaced into the main vhost ('structure' section)
```

(continues on next page)

(continued from previous page)

```

# into their corresponding position:
#
#   __XDOMAIN_REQ__
#   __ALIASES__
#   __DENIES__
#   __STATUS__
#
# The features itself also contain variables to be adjusted in conf.yml
# and will then be replaced in their corresponding feature section
# before being replaced into the vhost section (if enabled):
#
# XDomain:
#   __REGEX__
# Alias:
#   __REGEX__
#   __PATH__
# Deny:
#   __REGEX__
# Status:
#   __REGEX__
#
# Variables to be replaced directly in the vhost configuration can also be set
# in conf.yml and include:
#   __VHOST_NAME__
#   __DOCUMENT_ROOT__
#   __INDEX__
#   __ACCESS_LOG__
#   __ERROR_LOG__
#
###
### Basic vHost skeleton
###
vhost: |
    <VirtualHost __DEFAULT_VHOST__:__PORT__>
        ServerName   __VHOST_NAME__

        CustomLog    "__ACCESS_LOG__" combined
        ErrorLog     "__ERROR_LOG__"

        # Reverse Proxy definition (Ensure to adjust the port, currently '8000')
        ProxyRequests On
        ProxyPreserveHost On
        ProxyPass / http://php:8000/
        ProxyPassReverse / http://php:8000/

        __REDIRECT__
        __SSL__
        __ALIASES__
        __DENIES__
        __SERVER_STATUS__
        # Custom directives
        __CUSTOM__
    </VirtualHost>

###

```

(continues on next page)

(continued from previous page)

```

### vHost Type (normal or reverse proxy)
###
vhost_type:
  docroot: ""
  rproxy: ""

###
### Optional features to be enabled in vHost
###
features:

  # SSL Configuration
  ssl: |
    SSLEngine on
    SSLCertificateFile    "__SSL_PATH_CRT__"
    SSLCertificateKeyFile "__SSL_PATH_KEY__"
    SSLProtocol           __SSL_PROTOCOLS__
    SSLHonorCipherOrder   __SSL_HONOR_CIPHER_ORDER__
    SSLCipherSuite        __SSL_CIPHERS__

  # Redirect to SSL directive
  redirect: |
    RedirectMatch (.*) https://__VHOST_NAME__:__SSL_PORT__$1

  # PHP-FPM left empty, as we are an reverse proxy configuration
  php_fpm: ""

  alias: |
    # Alias Definition
    Alias "__ALIAS__" "__PATH__ALIAS__"
    <Location "__ALIAS__">
      __XDOMAIN_REQ__
    </Location>
    <Directory "__PATH__ALIAS__">
      Order allow,deny
      Allow from all
    </Directory>

  deny: |
    # Deny Definition
    <FilesMatch "__REGEX__">
      Order allow,deny
      Deny from all
    </FilesMatch>

  server_status: |
    # Status Page
    <Location __REGEX__>
      SetHandler server-status
      Order allow,deny
      Allow from all
    </Location>

  xdomain_request: |
    # Allow cross domain request from these hosts
    SetEnvIf Origin "__REGEX__" AccessControlAllowOrigin=$0
    Header add Access-Control-Allow-Origin %{AccessControlAllowOrigin}e_
    ↪env=AccessControlAllowOrigin

```

(continues on next page)

(continued from previous page)

```

Header always set Access-Control-Allow-Methods "POST, GET, OPTIONS, DELETE, PUT"
Header always set Access-Control-Max-Age "0"
Header always set Access-Control-Allow-Headers "x-requested-with, Content-Type,
↪origin, authorization, accept, client-security-token"
# Added a rewrite to respond with a 200 SUCCESS on every OPTIONS request.
RewriteEngine On
RewriteCond %{REQUEST_METHOD} OPTIONS
RewriteRule ^(.*)$ $1 [R=200,L]

```

### 33.3.2 Apache 2.4 template

Listing 5: apache24.yml-example-rproxy

```

---
# Apache 2.4 Reverse Proxy Template defintion for vhost-gen.py
#
# The 'feature' section contains optional features that can be enabled via
# conf.yml and will then be replaced into the main vhost ('structure' section)
# into their corresponding position:
#
#   __XDOMAIN_REQ__
#   __ALIASES__
#   __DENIES__
#   __STATUS__
#
# The features itself also contain variables to be adjusted in conf.yml
# and will then be replaced in their corresponding feature section
# before being replaced into the vhost section (if enabled):
#
# XDomain:
#   __REGEX__
# Alias:
#   __REGEX__
#   __PATH__
# Deny:
#   __REGEX__
# Status:
#   __REGEX__
#
# Variables to be replaced directly in the vhost configuration can also be set
# in conf.yml and include:
#   __VHOST_NAME__
#   __DOCUMENT_ROOT__
#   __INDEX__
#   __ACCESS_LOG__
#   __ERROR_LOG__
#
###
### Basic vHost skeleton
###
vhost: |
  <VirtualHost __DEFAULT_VHOST__:__PORT__>

```

(continues on next page)

(continued from previous page)

```

ServerName __VHOST_NAME__
Protocols __HTTP_PROTO__

CustomLog "__ACCESS_LOG__" combined
ErrorLog  "__ERROR_LOG__"

# Reverse Proxy definition (Ensure to adjust the port, currently '8000')
ProxyRequests On
ProxyPreserveHost On
ProxyPass / http://php:8000/
ProxyPassReverse / http://php:8000/

__REDIRECT__
__SSL__
__ALIASES__
__DENIES__
__SERVER_STATUS__
# Custom directives
__CUSTOM__
</VirtualHost>

###
### vHost Type (normal or reverse proxy)
###
vhost_type:
  docroot: ""
  rproxy: ""

###
### Optional features to be enabled in vHost
###
features:

# SSL Configuration
ssl: |
  SSLEngine on
  SSLCertificateFile  "__SSL_PATH_CRT__"
  SSLCertificateKeyFile  "__SSL_PATH_KEY__"
  SSLProtocol         __SSL_PROTOCOLS__
  SSLHonorCipherOrder  __SSL_HONOR_CIPHER_ORDER__
  SSLCipherSuite       __SSL_CIPHERS__

# Redirect to SSL directive
redirect: |
  RedirectMatch (.*) https://__VHOST_NAME__:__SSL_PORT__$1

# PHP-FPM left empty, as we are an reverse proxy configuration
php_fpm: ""

alias: |
# Alias Definition
Alias "__ALIAS__" "__PATH__ALIAS__"
<Location "__ALIAS__">
  __XDOMAIN_REQ__
</Location>
<Directory "__PATH__ALIAS__">
  Order allow,deny

```

(continues on next page)

(continued from previous page)

```

        Allow from all
        Require all granted
    </Directory>

deny: |
    # Deny Definition
    <FilesMatch "__REGEX__">
        Order allow,deny
        Deny from all
    </FilesMatch>

server_status: |
    # Status Page
    <Location __REGEX__>
        SetHandler server-status
        Order allow,deny
        Allow from all
        Require all granted
    </Location>

xdomain_request: |
    # Allow cross domain request from these hosts
    SetEnvIf Origin "__REGEX__" AccessControlAllowOrigin=$0
    Header add Access-Control-Allow-Origin %{AccessControlAllowOrigin}e
    ↪env=AccessControlAllowOrigin
    Header always set Access-Control-Allow-Methods "POST, GET, OPTIONS, DELETE, PUT"
    Header always set Access-Control-Max-Age "0"
    Header always set Access-Control-Allow-Headers "x-requested-with, Content-Type,
    ↪origin, authorization, accept, client-security-token"
    # Added a rewrite to respond with a 200 SUCCESS on every OPTIONS request.
    RewriteEngine On
    RewriteCond %{REQUEST_METHOD} OPTIONS
    RewriteRule ^(.*)$ $1 [R=200,L]

```

### 33.3.3 Nginx template

Listing 6: nginx.yml-example-rproxy

```

---

# Nginx Reverse Proxy Template defintion for vhost-gen.py
#
# The 'feature' section contains optional features that can be enabled via
# conf.yml and will then be replaced into the main vhost ('structure' section)
# into their corresponding position:
#
#   __XDOMAIN_REQ__
#   __ALIASES__
#   __DENIES__
#   __STATUS__
#
# The features itself also contain variables to be adjusted in conf.yml
# and will then be replaced in their corresponding feature section
# before being replaced into the vhost section (if enabled):
#

```

(continues on next page)

(continued from previous page)

```

# XDomain:
#   __REGEX__
# Alias:
#   __REGEX__
#   __PATH__
# Deny:
#   __REGEX__
# Status:
#   __REGEX__
#
# Variables to be replaced directly in the vhost configuration can also be set
# in conf.yml and include:
#   __VHOST_NAME__
#   __DOCUMENT_ROOT__
#   __INDEX__
#   __ACCESS_LOG__
#   __ERROR_LOG__
#
###
### Basic vHost skeleton
###
vhost: |
  server {
    listen      __PORT__ __HTTP_PROTO__ __DEFAULT_VHOST__;
    server_name __VHOST_NAME__;

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

    # Reverse Proxy definition (Ensure to adjust the port, currently '8000')
    location / {
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_pass http://php:8000;
    }

    __REDIRECT__
    __SSL__
    __ALIASES__
    __DENIES__
    __SERVER_STATUS__
    # Custom directives
    __CUSTOM__
  }

###
### vHost Type (normal or reverse proxy)
###
vhost_type:
  docroot: ""
  rproxy: ""

###
### Optional features to be enabled in vHost
###

```

(continues on next page)



(continued from previous page)

```

features:

# SSL Configuration
ssl: |
    ssl_certificate      __SSL_PATH_CRT__;
    ssl_certificate_key  __SSL_PATH_KEY__;
    ssl_protocols        __SSL_PROTOCOLS__;
    ssl_prefer_server_ciphers __SSL_HONOR_CIPHER_ORDER__;
    ssl_ciphers          __SSL_CIPHERS__;

# Redirect to SSL directive
redirect: |
    return 301 https://__VHOST_NAME__:__SSL_PORT__$request_uri;

# PHP-FPM left empty, as we are an reverse proxy configuration
php_fpm: ""

alias: |
    # Alias Definition
    location ~ __ALIAS__ {
        root __PATH__;
        __XDOMAIN_REQ__
    }

deny: |
    # Deny Definition
    location ~ __REGEX__ {
        deny all;
    }

server_status: |
    # Status Page
    location ~ __REGEX__ {
        stub_status on;
        access_log off;
    }

xdomain_request: |
    # Allow cross domain request from these hosts
    if ( $http_origin ~* (__REGEX__) ) {
        add_header "Access-Control-Allow-Origin" "$http_origin";
        add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
        add_header 'Access-Control-Allow-Headers' 'DNT,X-CustomHeader,Keep-Alive,User-
↪Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Content-Range,
↪Range';
        add_header 'Access-Control-Expose-Headers' 'DNT,X-CustomHeader,Keep-Alive,
↪User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Content-
↪Range,Range';
        add_header 'Access-Control-Max-Age' 0;
        return 200;
    }

```



---

## Customize all virtual hosts globally

---

### Table of Contents

- *Prerequisite*
- *Apply templates globally to all vhosts*
  - *Apache 2.2*
  - *Apache 2.4*
  - *Nginx stable and Nginx mainline*

### 34.1 Prerequisite

Ensure you have read and understood how vhost-gen templates work and where to find them

**See also:**

*Virtual host templates*

### 34.2 Apply templates globally to all vhosts

When applying those templates, you do it globally for all projects. The only exception is if you have already a specific vhost template for a project in place.

**See also:**

*Customize specific virtual host*

In order for template files to be picked up by the web server they must be copied to their correct filename.

Web server	Example template	Template name
Apache 2.2	apache22.yml-example-vhost	apache22.yml
Apache 2.4	apache24.yml-example-vhost	apache24.yml
Nginx stable	nginx.yml-example-vhost	nginx.yml
Nginx mainline	nginx.yml-example-vhost	nginx.yml

---

**Important:** Do not use \*.yml-example-rproxy templates for global configuration. These are only intended to be used on a per project base.

---

---

**Note:** If you simply copy the files to their corresponding template file name, nothing will change as those templates reflect the same values the web servers are using.

---

### 34.2.1 Apache 2.2

1. Navigate to `cfg/vhost-gen/` inside the Devilbox directory
2. Copy `apache22.yml-example-vhost` to `apache22.yml` and restart the Devilbox
3. Whenever you adjust `apache22.yml`, you need to restart the Devilbox

### 34.2.2 Apache 2.4

1. Navigate to `cfg/vhost-gen/` inside the Devilbox directory
2. Copy `apache24.yml-example-vhost` to `apache24.yml` and restart the Devilbox
3. Whenever you adjust `apache24.yml`, you need to restart the Devilbox

### 34.2.3 Nginx stable and Nginx mainline

1. Navigate to `cfg/vhost-gen/` inside the Devilbox directory
2. Copy `nginx.yml-example-vhost` to `nginx.yml` and restart the Devilbox
3. Whenever you adjust `nginx.yml`, you need to restart the Devilbox

---

### Customize specific virtual host

---

#### 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:

- 
- 
- 

### 35.1.2 Where do I find templates

The latest version of vhost-gen templates are shipped in the Devilbox git directory under `cfg/vhost-gen/`.

### 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 `cfg/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/cfg/vhost-gen
Project name	project-1
<code>HTTPD_TEMPLATE_DIR</code>	.devilbox (default value)
<code>HOST_PATH_HTTPD_DATADIR</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: `.devilbox`.

---

**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 cfg/vhost-gen/*.yaml-example ./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	apache22.yml
Apache 2.4	apache24.yml
Nginx stable	nginx.yml
Nginx mainline	nginx.yml

---

**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

# Check the syntax of apache22.yml
devilbox@php-7.0.19 in /shared/httpd/project-1/.devilbox $ yamllint apache22.yml
```

(continues on next page)



(continued from previous page)

```

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:

- *Virtual host templates*
- *Example: add sub domains*

---

### Virtual host vs Reverse Proxy

---

---

**Note:** Ensure you have read *Virtual host templates*

---

#### Table of Contents

- *Motivation*
- *Benefits*
- *Creating a reverse proxy*

### 36.1 Motivation

By default, all virtual hosts will simply serve files located in your document root directory within your project directory. Sometimes however your *project* is already its own server that will serve requests through a listening network port. (e.g. a running NodeJS application). This listening port will however only be available inside the PHP container (or any other container) you have added to the Devilbox and the webserver is not aware of this.

For this kind of project you will want to create a reverse proxy which simply forwards the requests incoming to the webserver to your application (either to the PHP container to a specific port or to any other container you have attached).

### 36.2 Benefits

By using the already available web server to reverse proxy requests to your service you will be able to have all the current features for you application such as:

- Have it displayed in the intranet page

- Have standardized domain names
- Have valid HTTPS

## 36.3 Creating a reverse proxy

Creating a reverse proxy is as simply as copying the `vhost-gen` templates to your project directory.

In order to make your life simple there are a few generic docs that get you started and let you know more about the theory behind it:

**See also:**

- *[Reverse Proxy with HTTPS](#)*
- *[Reverse Proxy for custom Docker](#)*

If this is too generic you can also have a look at two specific examples to setup fully automated Reverse Proxies including autostarting your application on `docker-compose up`.

**See also:**

- *[Setup reverse proxy NodeJS](#)*
- *[Setup reverse proxy Sphinx docs](#)*

---

### Example: add 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

- *Simple sub domains for one project*
  - *Example*
    - \* *Prerequisite*
    - \* *Directory structure*
- *Complex 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*

## 37.1 Simple sub domains for one project

When you just want to serve your project under different sub domains, 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	local	http://my-test.local
www.my-test	local	http://www.my-test.local
t2.www.my-test	local	http://t2.www.my-test.local

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>

---

**Important:** The project directories must be real directories and not symlinks! See example below for how to set it up.

---

### 37.1.1 Example

#### Prerequisite

Let's assume the following settings.



Variable	Value
<code>TLD_SUFFIX</code>	<code>loc</code>
<code>HTTPD_DOCROOT_DIR</code>	<code>htdocs</code>
Project name / directory	<code>my-test</code>
Sub domain 1 / directory	<code>api.my-test</code>
Sub domain 2 / directory	<code>www.my-test</code>

- Project which holds the data is `my-test`
- Web root of `my-test` is in `my-test/Framework/public`
- Same project should be available under `api.my-test` and `www.my-test`

## Directory structure

```
host> tree -L 2
.
├── my-test
│   └── Framework
│       └── htdocs -> Framework/public
├── api.my-test
│   └── htdocs -> ../my-test/Framework/public
└── www.my-test
    └── htdocs -> ../my-test/Framework/public
```

- `my-test`, `api.my-test` and `www.my-test` must be **normal directories** (not symlinks).
- The *framework data* resided in `my-test`.
- Each projects `htdocs` directory is a symlink pointing to the web root of the *framework* in `my-test`

With this structure you will have three domains available pointing to the same project:

- <http://my-test.loc>
- <http://api.my-test.loc>
- <http://www.my-test.loc>

## 37.2 Complex sub domains for one project

When you want to have more complex 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` 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:

**Virtual host templates** Ensure you have read and understand how to customize virtual hosts globally with `vhost-gen`.

**Customize all virtual hosts globally** Ensure you have read and understand how to customize virtual hosts globally with `vhost-gen`.

*Customize specific virtual host* 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 PHP container](#)
- [Available tools](#)

### 37.2.1 Prerequisite

Let's assume the following settings.

Variable	Value
Devilbox path	<code>/home/user/devilbox</code>
<i>HTTPD_TEMPLATE_DIR</i>	<code>.devilbox</code>
<i>HOST_PATH_HTTPD_DATADIR</i>	<code>./data/www</code>
<i>TLD_SUFFIX</i>	<code>loc</code>
Project name/directory	<code>project-1</code> (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 cfg/vhost-gen/*.yaml ./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.

## 37.2.2 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 a “`ServerAlias`” directive:

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

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

    CustomLog "__ACCESS_LOG__" combined
    ErrorLog  "__ERROR_LOG__"

    __VHOST_DOCROOT__
    __VHOST_RPROXY__
```

(continues on next page)

(continued from previous page)

```

__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: [Setup 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

```

(continues on next page)

(continued from previous page)

```
__CUSTOM__
</VirtualHost>
```

All you will have to do, is to add a `ServerAlias` 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__
    ServerAlias *.__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.

## 37.2.3 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](#)

## 37.2.4 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: [Setup 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 {
```

(continues on next page)

(continued from previous page)

```

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.

#### 37.2.5 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.

## 37.2.6 Checklist

1. Template files are copied from `cfg/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



---

### Reverse Proxy with HTTPS

---

Imagine you have started an application within the PHP container that creates a listening port (e.g.: NodeJS). This will now only listen on the PHP container and you would have to adjust the docker-compose.yml definition in order to have that port available outside to your host OS.

Alternatively, there is a simple way to reverse proxy it to the already running web server and even make use of the available HTTPS feature.

**See also:**

**Read more about how to autostart applications:**

- *Custom scripts per PHP version* (individually for different PHP versions)
- *Custom scripts globally* (equal for all PHP versions)

#### Table of Contents

- *Walkthrough*
  - *Assumption*
  - *Copy vhost-gen templates*
  - *Adjust port*
    - \* *Adjust Apache 2.2 template*
    - \* *Adjust Apache 2.4 template*
    - \* *Adjust Nginx template*
  - *Restart the Devilbox*
  - *Start your application*
  - *Visit your project*

## 38.1 Walkthrough

### 38.1.1 Assumption

Let's imagine you have started an application in the PHP container with the following settings:

- `TLD_SUFFIX`: `loc`
- *Project directory* inside PHP container: `/shared/httpd/my-app`
- `HOST_PATH_HTTPD_DATADIR` on the host: `./data/www`
- `HTTPD_TEMPLATE_DIR`: `.devilbox`
- Listening port: `8081`
- Listening host: `php` (any of the PHP container)
- Resulting vhost name: `my-app.loc`

### 38.1.2 Copy vhost-gen templates

The reverse vhost-gen templates are available in `cfg/vhost-gen`:

```
host> tree -L 1 cfg/vhost-gen/

cfg/vhost-gen/
├── apache22.yml-example-rproxy
├── apache22.yml-example-vhost
├── apache24.yml-example-rproxy
├── apache24.yml-example-vhost
├── nginx.yml-example-rproxy
├── nginx.yml-example-vhost
└── README.md

0 directories, 7 files
```

For this example we will copy all `*-example-rproxy` files into `/shared/httpd/my-app/.devilbox` to ensure this will work with all web servers.

```
host> cd /path/to/devilbox
host> cp cfg/vhost-gen/apache22.yml-example-rproxy data/www/my-app/.devilbox/apache22.
↪yml
host> cp cfg/vhost-gen/apache24.yml-example-rproxy data/www/my-app/.devilbox/apache24.
↪yml
host> cp cfg/vhost-gen/nginx.yml-example-rproxy data/www/my-app/.devilbox/nginx.yml
```

### 38.1.3 Adjust port

By default, all vhost-gen templates will forward requests to port `8000` into the PHP container. Our current example however uses port `8081`, so we must change that accordingly for all three templates.

#### Adjust Apache 2.2 template

Open the `apache22.yml` vhost-gen template in your project:

```
host> cd /path/to/devilbox
host> vi data/www/my-app/.devilbox/apache22.yml
```

Find the two lines with ProxyPass and ProxyPassReverse and change the port from 8000 to 8081

Listing 1: data/www/my-app/.devilbox/apache22.yml

```
# ... more lines above ... #

###
### Basic vHost skeleton
###
vhost: |
  <VirtualHost __DEFAULT_VHOST__:__PORT__>
    ServerName __VHOST_NAME__

    CustomLog "__ACCESS_LOG__" combined
    ErrorLog  "__ERROR_LOG__"

    # Reverse Proxy definition (Ensure to adjust the port, currently '8000')
    ProxyRequests On
    ProxyPreserveHost On
    ProxyPass / http://php:8081/
    ProxyPassReverse / http://php:8081/

# ... more lines below ... #
```

## Adjust Apache 2.4 template

Open the apache24.yml vhost-gen template in your project:

```
host> cd /path/to/devilbox
host> vi data/www/my-app/.devilbox/apache24.yml
```

Find the two lines with ProxyPass and ProxyPassReverse and change the port from 8000 to 8081

Listing 2: data/www/my-app/.devilbox/apache24.yml

```
# ... more lines above ... #

###
### Basic vHost skeleton
###
vhost: |
  <VirtualHost __DEFAULT_VHOST__:__PORT__>
    ServerName __VHOST_NAME__

    CustomLog "__ACCESS_LOG__" combined
    ErrorLog  "__ERROR_LOG__"

    # Reverse Proxy definition (Ensure to adjust the port, currently '8000')
    ProxyRequests On
    ProxyPreserveHost On
    ProxyPass / http://php:8081/
    ProxyPassReverse / http://php:8081/
```

(continues on next page)

(continued from previous page)

```
# ... more lines below ... #
```

## Adjust Nginx template

Open the `nginx.yml` vhost-gen template in your project:

```
host> cd /path/to/devilbox
host> vi data/www/my-app/.devilbox/nginx.yml
```

Find the lines with `proxy_pass` and change the port from 8000 to 8081

Listing 3: `data/www/my-app/.devilbox/nginx.yml`

```
# ... more lines above ... #

###
### Basic vHost skeleton
###
vhost: |
  server {
    listen      __PORT__DEFAULT_VHOST__;
    server_name __VHOST_NAME__;

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

    # Reverse Proxy definition (Ensure to adjust the port, currently '8000')
    location / {
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_pass http://php:8081;
    }

# ... more lines below ... #
```

### 38.1.4 Restart the Devilbox

Now for the changes to take affect, simply restart the Devilbox (or start if not yet running):

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

# Stop the Devilbox
host> docker-compose stop
host> docker-compose rm -f

# Start the Devilbox (PHP and HTTPD container only)
host> docker-compose up -d php httpd bind
```

### 38.1.5 Start your application

Enter the PHP container and start your application which creates the listening port in port 8081.

**See also:**

This can also be automated to happen automatically during `docker-compose up` via:

- *Custom scripts per PHP version* (individually for different PHP versions)
- *Custom scripts globally* (equal for all PHP versions)
- Example: *Autostarting NodeJS Apps*

### 38.1.6 Visit your project

That's it, your service application will now be available via:

- <http://my-app.loc>

It will also be available on HTTPS. This is by default and automatically:

- <https://my-app.loc>

**See also:**

*Setup valid HTTPS*

And is even shown as a project in the Devilbox intranet:

- <http://localhost/vhosts.php>
- <https://localhost/vhosts.php>





---

### Reverse Proxy for custom Docker

---

Imagine you have added a custom service container to the Devilbox which has a project that is available via http on a very specific port in that container.

You do not want to expose this port to the host system, but rather want to make it available via the bundled web server and also be able to see it on the Devilbox intranet vhost section.

Additionally you want the project to make use of the same DNS naming scheme and also have HTTPS for it.

You can easily achieve this by setting up a reverse proxy for it.

**See also:**

*Add your own Docker image*

**Table of Contents**

- *Walkthrough*
  - *Assumption*
  - *Create virtual directory*
  - *Copy vhost-gen templates*
  - *Adjust port*
    - \* *Adjust Apache 2.2 template*
    - \* *Adjust Apache 2.4 template*
    - \* *Adjust Nginx template*
  - *Restart the Devilbox*
  - *Visit your project*

## 39.1 Walkthrough

### 39.1.1 Assumption

Let's imagine you have added a custom Python Docker image to the Devilbox which starts up a Django application listening on port 3000.

- *TLD\_SUFFIX*: loc
- Desired DNS name: my-python.loc
- *HOST\_PATH\_HTTPD\_DATADIR* on the host: ./data/www
- *HTTPD\_TEMPLATE\_DIR*: .devilbox
- Listening port: 3000
- Listening host: python (hostname of the Python container)

### 39.1.2 Create *virtual* directory

In order to create a reverse proxy for that custom container, you must add a *virtual* project directory without any data in it. This directory is purely for the purpose of determining the DNS name and for having the vhost-gen configuration in.

Navigate to the *HOST\_PATH\_HTTPD\_DATADIR* directory and create your project

```
host> cd /path/to/devilbox
host> cd /path/to/devilbox/data/www

# Create the project directory
host> mkdir my-python

# Create the htdocs directory (to have a valid project for the Intranet page)
host> mkdir my-python/htdocs

# Create the vhost-gen directory (to be apply to apply custom templates)
host> mkdir my-python/.devilbox
```

This part is now sufficient to have the project visible on the Devilbox intranet.

### 39.1.3 Copy vhost-gen templates

The reverse vhost-gen templates are available in `cfg/vhost-gen`:

```
host> tree -L 1 cfg/vhost-gen/

cfg/vhost-gen/
├── apache22.yml-example-rproxy
├── apache22.yml-example-vhost
├── apache24.yml-example-rproxy
├── apache24.yml-example-vhost
├── nginx.yml-example-rproxy
├── nginx.yml-example-vhost
└── README.md
```

(continues on next page)

(continued from previous page)

```
0 directories, 7 files
```

For this example we will copy all `*-example-rproxy` files into `/shared/httpd/my-python/.devilbox` to ensure this will work with all web servers.

```
host> cd /path/to/devilbox
host> cp cfg/vhost-gen/apache22.yml-example-rproxy data/www/my-python/.devilbox/
↪apache22.yml
host> cp cfg/vhost-gen/apache24.yml-example-rproxy data/www/my-python/.devilbox/
↪apache24.yml
host> cp cfg/vhost-gen/nginx.yml-example-rproxy data/www/my-python/.devilbox/nginx.yml
```

### 39.1.4 Adjust port

By default, all vhost-gen templates will forward requests to port 8000 into the PHP container. Our current example however uses port 3000 and host python, so we must change that accordingly for all three templates.

#### Adjust Apache 2.2 template

Open the `apache22.yml` vhost-gen template in your project:

```
host> cd /path/to/devilbox
host> vi data/www/my-python/.devilbox/apache22.yml
```

Find the two lines with `ProxyPass` and `ProxyPassReverse` and change the port from 8000 to 3000 and host php to python:

Listing 1: `data/www/my-python/.devilbox/apache22.yml`

```
# ... more lines above ... #

###
### Basic vHost skeleton
###
vhost: |
  <VirtualHost __DEFAULT_VHOST__:__PORT__>
    ServerName  __VHOST_NAME__

    CustomLog   "__ACCESS_LOG__" combined
    ErrorLog    "__ERROR_LOG__"

    # Reverse Proxy definition (Ensure to adjust the port, currently '8000')
    ProxyRequests On
    ProxyPreserveHost On
    ProxyPass   / http://python:3000/
    ProxyPassReverse / http://python:3000/

# ... more lines below ... #
```

#### Adjust Apache 2.4 template

Open the `apache24.yml` vhost-gen template in your project:

```
host> cd /path/to/devilbox
host> vi data/www/my-python/.devilbox/apache24.yml
```

Find the two lines with ProxyPass and ProxyPassReverse and change the port from 8000 to 3000 and host php to python:

Listing 2: data/www/my-python/.devilbox/apache24.yml

```
# ... more lines above ... #

###
### Basic vHost skeleton
###
vhost: |
    <VirtualHost __DEFAULT_VHOST__:__PORT__>
        ServerName    __VHOST_NAME__

        CustomLog      "__ACCESS_LOG__" combined
        ErrorLog        "__ERROR_LOG__"

        # Reverse Proxy definition (Ensure to adjust the port, currently '8000')
        ProxyRequests On
        ProxyPreserveHost On
        ProxyPass / http://python:3000/
        ProxyPassReverse / http://python:3000/

# ... more lines below ... #
```

## Adjust Nginx template

Open the nginx.yml vhost-gen template in your project:

```
host> cd /path/to/devilbox
host> vi data/www/my-python/.devilbox/nginx.yml
```

Find the line with proxy\_pass and change the port from 8000 to 3000 and host php to python:

Listing 3: data/www/my-python/.devilbox/nginx.yml

```
# ... more lines above ... #

###
### Basic vHost skeleton
###
vhost: |
    server {
        listen          __PORT__ __DEFAULT_VHOST__;
        server_name     __VHOST_NAME__;

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

        # Reverse Proxy definition (Ensure to adjust the port, currently '8000')
        location / {
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
```

(continues on next page)

(continued from previous page)

```
    proxy_pass http://python:3000;
  }

# ... more lines below ... #
```

### 39.1.5 Restart the Devilbox

Now for the changes to take affect, simply restart the Devilbox (or start if not yet running):

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

# Stop the Devilbox
host> docker-compose stop
host> docker-compose rm -f

# Start the Devilbox (Your Python container and the PHP and HTTPD container only)
host> docker-compose up -d php httpd bind python
```

### 39.1.6 Visit your project

That's it, your service application will now be available via:

- <http://my-python.loc>

It will also be available on HTTPS. This is by default and automatically:

- <https://my-python.loc>

**See also:**

*Setup valid HTTPS*

And is even shown as a project in the Devilbox intranet:

- <http://localhost/vhosts.php>
- <https://localhost/vhosts.php>



---

## Enable all additional container

---

Besides providing basic LAMP/MEAN stack container, which are well integrated into the Devilbox intranet, the Devilbox also ships additional pre-configured container that can easily be enabled.

### Table of Contents

- *Available additional container*
- *Enable all additional container*
- *Configure additional container*

## 40.1 Available additional container

The following table shows you the currently additional available container:

Container	Name	Hostname	IP Address
Blackfire	blackfire	blackfire	172.16.238.200
MailHog	mailhog	mailhog	172.16.238.201
RabbitMQ	rabbit	rabbit	172.16.238.210
Solr	solr	solr	172.16.238.220

## 40.2 Enable all additional container

Copy `docker-compose.override.yml-all` into the root of the Devilbox git directory.

```
host> cp compose/docker-compose.override.yml-all docker-compose.override.yml
```

That's it, if you `docker-compose up`, all container will be started. This however is not advised as it will eat up a lot of resources. You are better off by selectively specifying the container you want to run.

**See also:**

*Start the Devilbox*

## 40.3 Configure additional container

The additional container also provide many configuration options just as the default ones do. That includes, but is not limited to:

- Image version
- Exposed ports
- Mount points
- And various container specific settings

In order to fully customize each container, refer to their own documentation section:

**See also:**

- *Enable and configure Blackfire*
- *Enable and configure MailHog*
- *Enable and configure RabbitMQ*
- *Enable and configure Solr*



---

## Enable and configure Blackfire

---

This section will guide you through getting Blackfire integrated into the Devilbox.

**See also:**

- 
- 
- *Enable all additional container*

**Table of Contents**

- *Overview*
  - *Available overwrites*
  - *Blackfire settings*
  - *Blackfire env variables*
- *Instructions*
  - *1. Copy docker-compose.override.yml*
  - *2. Adjust env settings (required)*
  - *3. Start the Devilbox*
- *TL;DR*

## 41.1 Overview

### 41.1.1 Available overwrites

The Devilbox ships various example configurations to overwrite the default stack. Those files are located under `compose/` in the Devilbox git directory.

`docker-compose.override.yml-all` has all examples combined in one file for easy copy/paste. However, each example also exists in its standalone file as shown below:

```
host> tree -L 1 compose/
compose/
├── docker-compose.override.yml-all
├── docker-compose.override.yml-blackfire
├── docker-compose.override.yml-mailhog
├── docker-compose.override.yml-rabbitmq
├── docker-compose.override.yml-solr
└── README.md

0 directories, 6 files
```

**See also:**

*[Enable all additional container](#)*

### 41.1.2 Blackfire settings

In case of Blackfire, the file is `compose/docker-compose.override.yml-blackfire`. This file must be copied into the root of the Devilbox git directory.

What	How and where		
Example compose file	<code>compose/docker-compose.override.yml-all</code>	or	<code>compose/docker-compose.override.yml-blackfire</code>
Container IP address	<code>172.16.238.200</code>		
Container host name	<code>blackfire</code>		
Container name	<code>blackfire</code>		
Mount points	none		
Exposed port	none		
Available at	n.a.		
Further configuration	<code>BLACKFIRE_SERVER_ID</code> and <code>BLACKFIRE_SERVER_TOKEN</code> must be set via <code>.env</code>		

### 41.1.3 Blackfire env variables

Additionally the following `.env` variables can be created for easy configuration:

Variable	Default value	Description
<code>BLACKFIRE_SERVER_ID</code>	<code>id</code>	A valid server id is required in order to use blackfire.
<code>BLACKFIRE_SERVER_TOKEN</code>	<code>token</code>	A valid server token is required in order to use blackfire.
<code>BLACKFIRE_SERVER</code>	<code>latest</code>	Controls the Blackfire version to use.

## 41.2 Instructions

### 41.2.1 1. Copy docker-compose.override.yml

Copy the Blackfire Docker Compose overwrite file into the root of the Devilbox git directory. (It must be at the same level as the default `docker-compose.yml` file).

```
host> cp compose/docker-compose.override.yml-blackfire docker-compose.override.yml
```

See also:

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

### 41.2.2 2. Adjust env settings (required)

By default Blackfire is using some dummy values for `BLACKFIRE_SERVER_ID` and `BLACKFIRE_SERVER_TOKEN`. You must however acquire valid values and set them in your `.env` file in order for Blackfire to properly start. Those values can be obtained at their official webpage.

Listing 1: `.env`

```
BLACKFIRE_SERVER_ID=<valid server id>
BLACKFIRE_SERVER_TOKEN=<valid server token>

#BLACKFIRE_SERVER=1.12.0
#BLACKFIRE_SERVER=1.13.0
#BLACKFIRE_SERVER=1.14.0
#BLACKFIRE_SERVER=1.14.1
#BLACKFIRE_SERVER=1.15.0
#BLACKFIRE_SERVER=1.16.0
#BLACKFIRE_SERVER=1.17.0
#BLACKFIRE_SERVER=1.17.1
#BLACKFIRE_SERVER=1.18.0
BLACKFIRE_SERVER=latest
```

See also:

*.env file*

### 41.2.3 3. Start the Devilbox

The final step is to start the Devilbox with Blackfire.

Let's assume you want to start php, httpd, bind and blackfire.

```
host> docker-compose up -d php httpd bind blackfire
```

See also:

*Start the Devilbox*

## 41.3 TL;DR

For the lazy readers, here are all commands required to get you started. Simply copy and paste the following block into your terminal from the root of your Devilbox git directory:

```
# Copy compose-override.yml into place
cp compose/docker-compose.override.yml-blackfire docker-compose.override.yml

# Create .env variable
echo "BLACKFIRE_SERVER_ID=<valid server id>" >> .env
echo "BLACKFIRE_SERVER_TOKEN=<valid server token>" >> .env

echo "#BLACKFIRE_SERVER=1.12.0" >> .env
echo "#BLACKFIRE_SERVER=1.13.0" >> .env
echo "#BLACKFIRE_SERVER=1.14.0" >> .env
echo "#BLACKFIRE_SERVER=1.14.1" >> .env
echo "#BLACKFIRE_SERVER=1.15.0" >> .env
echo "#BLACKFIRE_SERVER=1.16.0" >> .env
echo "#BLACKFIRE_SERVER=1.17.0" >> .env
echo "#BLACKFIRE_SERVER=1.17.1" >> .env
echo "#BLACKFIRE_SERVER=1.18.0" >> .env
echo "BLACKFIRE_SERVER=latest" >> .env

# Start container
docker-compose up -d php httpd bind blackfire
```

---

### Enable and configure MailHog

---

This section will guide you through getting MailHog integrated into the Devilbox.

**See also:**

- 
- 
- *Enable all additional container*

**Table of Contents**

- *Overview*
  - *Available overwrites*
  - *MailHog settings*
  - *MailHog env variables*
- *Instructions*
  - *1. Copy docker-compose.override.yml*
  - *2. Adjust PHP settings*
  - *3. Adjust .env settings (optional)*
  - *4. Start the Devilbox*
  - *5. Start using it*
- *TL;DR*

## 42.1 Overview

### 42.1.1 Available overwrites

The Devilbox ships various example configurations to overwrite the default stack. Those files are located under `compose/` in the Devilbox git directory.

`docker-compose.override.yml-all` has all examples combined in one file for easy copy/paste. However, each example also exists in its standalone file as shown below:

```
host> tree -L 1 compose/
compose/
├── docker-compose.override.yml-all
├── docker-compose.override.yml-blackfire
├── docker-compose.override.yml-mailhog
├── docker-compose.override.yml-rabbitmq
├── docker-compose.override.yml-solr
└── README.md

0 directories, 6 files
```

**See also:**

*[Enable all additional container](#)*

### 42.1.2 MailHog settings

In case of MailHog, the file is `compose/docker-compose.override.yml-mailhog`. This file must be copied into the root of the Devilbox git directory.

What	How and where		
Example compose file	<code>compose/docker-compose.override.yml-all</code>	or	<code>compose/docker-compose.override.yml-mailhog</code>
Container IP address	<code>172.16.238.201</code>		
Container host name	<code>mailhog</code>		
Container name	<code>mailhog</code>		
Mount points	none		
Exposed port	<code>8025</code> (can be changed via <code>.env</code> )		
Available at	<code>http://localhost:8025</code>		
Further configuration	php.ini settings need to be applied per version		

### 42.1.3 MailHog env variables

Additionally the following `.env` variables can be created for easy configuration:

Variable	Default value	Description
<code>HOST_PORT_MAILHOG</code>	<code>8025</code>	Controls the host port on which MailHog will be available at.
<code>MAILHOG_SERVER</code>	<code>latest</code>	Controls the MailHog version to use.

## 42.2 Instructions

### 42.2.1 1. Copy docker-compose.override.yml

Copy the MailHog Docker Compose overwrite file into the root of the Devilbox git directory. (It must be at the same level as the default `docker-compose.yml` file).

```
host> cp compose/docker-compose.override.yml-mailhog docker-compose.override.yml
```

See also:

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

### 42.2.2 2. Adjust PHP settings

The next step is to tell PHP that it should use a different mail forwarder.

Let's assume you are using PHP 7.2.

```
# Navigate to the PHP ini configuration directory of your chosen version
host> cd cfg/php-ini-7.2

# Create and open a new *.ini file
host> vi mailhog.ini
```

Add the following content to the newly created ini file:

Listing 1: mailhog.ini

```
[mail function]
sendmail_path = '/usr/local/bin/mhsendmail --smtp-addr="mailhog:1025"'
```

See also:

*php.ini*

### 42.2.3 3. Adjust .env settings (optional)

By Default MailHog is using the host port 8025, this can be adjusted in the `.env` file. Add `HOST_PORT__MAILHOG` to `.env` and customize its value.

Additionally also the MailHog version can be controlled via `MAILHOG_SERVER`.

Listing 2: .env

```
HOST_PORT__MAILHOG=8025
MAILHOG_SERVER=latest
```

See also:

*.env file*

### 42.2.4 4. Start the Devilbox

The final step is to start the Devilbox with MailHog.

Let's assume you want to start php, httpd, bind and mailhog.

```
host> docker-compose up -d php httpd bind mailhog
```

**See also:**

*Start the Devilbox*

### 42.2.5 5. Start using it

- Once the Devilbox is running, visit <http://localhost:8025> in your browser.
- Any email send by any of the Devilbox managed projects will then appear in MailHog

## 42.3 TL;DR

For the lazy readers, here are all commands required to get you started. Simply copy and paste the following block into your terminal from the root of your Devilbox git directory:

```
# Copy compose-override.yml into place
cp compose/docker-compose.override.yml-mailhog docker-compose.override.yml

# Create php.ini
echo "[mail function]" > cfg/php-ini-7.2/mailhog.ini
echo "sendmail_path = '/usr/local/bin/mhsendmail --smtp-addr=\"mailhog:1025\"'" >>
↪cfg/php-ini-7.2/mailhog.ini

# Create .env variable
echo "HOST_PORT_MAILHOG=8025" >> .env
echo "MAILHOG_SERVER=latest" >> .env

# Start container
docker-compose up -d php httpd bind mailhog
```



---

### Enable and configure RabbitMQ

---

This section will guide you through getting RabbitMQ integrated into the Devilbox.

**See also:**

- 
- 
- *Enable all additional container*

**Table of Contents**

- *Overview*
  - *Available overwrites*
  - *RabbitMQ settings*
  - *RabbitMQ env variables*
- *Instructions*
  - *1. Copy docker-compose.override.yml*
  - *2. Adjust .env settings (optional)*
  - *4. Start the Devilbox*
- *TL;DR*

## 43.1 Overview

### 43.1.1 Available overwrites

The Devilbox ships various example configurations to overwrite the default stack. Those files are located under `compose/` in the Devilbox git directory.

`docker-compose.override.yml-all` has all examples combined in one file for easy copy/paste. However, each example also exists in its standalone file as shown below:

```
host> tree -L 1 compose/
compose/
├── docker-compose.override.yml-all
├── docker-compose.override.yml-blackfire
├── docker-compose.override.yml-mailhog
├── docker-compose.override.yml-rabbitmq
├── docker-compose.override.yml-solr
└── README.md

0 directories, 6 files
```

**See also:**

*[Enable all additional container](#)*

### 43.1.2 RabbitMQ settings

In case of RabbitMQ, the file is `compose/docker-compose.override.yml-rabbitmq`. This file must be copied into the root of the Devilbox git directory.

What	How and where		
Example compose file	<code>compose/docker-compose.override.yml-all</code>	or	<code>compose/docker-compose.override.yml-rabbitmq</code>
Container IP address	172.16.238.210		
Container host name	rabbit		
Container name	rabbit		
Mount points	./data/rabbit (can be changed via <code>.env</code> )		
Exposed port	5672 and 15672 (can be changed via <code>.env</code> )		
Available at	<code>http://localhost:15672</code> (Admin WebUI)		
Further configuration	none		

### 43.1.3 RabbitMQ env variables

Additionally the following `.env` variables can be created for easy configuration:

Variable	Default value	Description
HOST_PORT_RABBIT	5672	Controls the host port on which RabbitMQ API will be available at.
HOST_PORT_RABBIT_MGMT	15672	Controls the host port on which RabbitMQ Admin WebUI will be available at.
RABBIT_SERVER	management	Controls the RabbitMQ version to use.
HOST_PATH_RABBIT_DATA	data/rabbit	Default mount point for persistent data.
RABBIT_DEFAULT_VHOST	my_vhost	Default RabbitMQ vhost name. (not a webserver vhost name)
RABBIT_DEFAULT_USER	guest	Default username for Admin WebUI.
RABBIT_DEFAULT_PASS	guest	Default password for Admin WebUI.

## 43.2 Instructions

### 43.2.1 1. Copy docker-compose.override.yml

Copy the RabbitMQ Docker Compose overwrite file into the root of the Devilbox git directory. (It must be at the same level as the default `docker-compose.yml` file).

```
host> cp compose/docker-compose.override.yml-rabbitmq docker-compose.override.yml
```

See also:

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

### 43.2.2 2. Adjust .env settings (optional)

RabbitMQ is using sane defaults, which can be changed by adding variables to the `.env` file and assigning custom values.

Add the following variables to `.env` and adjust them to your needs:

Listing 1: `.env`

```
# RabbitMQ version to choose
#RABBIT_SERVER=3.6
#RABBIT_SERVER=3.6-management
#RABBIT_SERVER=3.7
#RABBIT_SERVER=3.7-management
#RABBIT_SERVER=latest
RABBIT_SERVER=management

RABBIT_DEFAULT_VHOST=my_vhost
RABBIT_DEFAULT_USER=guest
RABBIT_DEFAULT_PASS=guest

HOST_PORT_RABBIT=5672
HOST_PORT_RABBIT_MGMT=15672
HOST_PATH_RABBIT_DATA=./data/rabbit
```

**See also:**

*.env file*

### 43.2.3 4. Start the Devilbox

The final step is to start the Devilbox with RabbitMQ.

Let's assume you want to start php, httpd, bind, rabbit.

```
host> docker-compose up -d php httpd bind rabbitmq
```

**See also:**

*Start the Devilbox*

## 43.3 TL;DR

For the lazy readers, here are all commands required to get you started. Simply copy and paste the following block into your terminal from the root of your Devilbox git directory:

```
# Copy compose-override.yml into place
cp compose/docker-compose.override.yml-rabbitmq docker-compose.override.yml

# Create .env variable
echo "# RabbitMQ version to choose" >> .env
echo "#RABBIT_SERVER=3.6" >> .env
echo "#RABBIT_SERVER=3.6-management" >> .env
echo "#RABBIT_SERVER=3.7" >> .env
echo "#RABBIT_SERVER=3.7-management" >> .env
echo "#RABBIT_SERVER=latest" >> .env
echo "RABBIT_SERVER=management" >> .env
echo "RABBIT_DEFAULT_VHOST=my_vhost" >> .env
echo "RABBIT_DEFAULT_USER=guest" >> .env
echo "RABBIT_DEFAULT_PASS=guest" >> .env
echo "HOST_PORT_RABBIT=5672" >> .env
echo "HOST_PORT_RABBIT_MGMT=15672" >> .env
echo "HOST_PATH_RABBIT_DATADIR=./data/rabbit" >> .env

# Start container
docker-compose up -d php httpd bind rabbit
```

---

### Enable and configure Solr

---

This section will guide you through getting Solr integrated into the Devilbox.

**See also:**

- 
- 
- *Enable all additional container*

**Table of Contents**

- *Overview*
  - *Available overwrites*
  - *Solr settings*
  - *Solr env variables*
- *Instructions*
  - *1. Copy docker-compose.override.yml*
  - *2. Adjust .env settings (optional)*
  - *4. Start the Devilbox*
- *TL;DR*

## 44.1 Overview

### 44.1.1 Available overwrites

The Devilbox ships various example configurations to overwrite the default stack. Those files are located under `compose/` in the Devilbox git directory.

`docker-compose.override.yml-all` has all examples combined in one file for easy copy/paste. However, each example also exists in its standalone file as shown below:

```
host> tree -L 1 compose/
compose/
├── docker-compose.override.yml-all
├── docker-compose.override.yml-blackfire
├── docker-compose.override.yml-mailhog
├── docker-compose.override.yml-rabbitmq
├── docker-compose.override.yml-solr
└── README.md

0 directories, 6 files
```

**See also:**

*[Enable all additional container](#)*

### 44.1.2 Solr settings

In case of Solr, the file is `compose/docker-compose.override.yml-solr`. This file must be copied into the root of the Devilbox git directory.

What	How and where	
Example compose file	<code>compose/docker-compose.override.yml-all</code>	or <code>compose/docker-compose.override.yml-solr</code>
Container IP address	172.16.238.220	
Container host name	solr	
Container name	solr	
Mount points	none	
Exposed port	8983 (can be changed via <code>.env</code> )	
Available at	<code>http://localhost:8983</code> (API and Admin WebUI)	
Further configuration	none	

### 44.1.3 Solr env variables

Additionally the following `.env` variables can be created for easy configuration:

Variable	Default value	Description
<code>HOST_PORT_SOLR</code>	8983	Controls the host port on which Solr API and WebUI will be available at.
<code>SOLR_SERVER</code>	latest	Controls the Solr version to use.
<code>SOLR_CORE_NAME</code>	devilbox	Default Solr core name

## 44.2 Instructions

### 44.2.1 1. Copy docker-compose.override.yml

Copy the Solr Docker Compose overwrite file into the root of the Devilbox git directory. (It must be at the same level as the default `docker-compose.yml` file).

```
host> cp compose/docker-compose.override.yml-solr docker-compose.override.yml
```

See also:

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

### 44.2.2 2. Adjust .env settings (optional)

Solr is using sane defaults, which can be changed by adding variables to the `.env` file and assigning custom values.

Add the following variables to `.env` and adjust them to your needs:

Listing 1: `.env`

```
# Solr version to choose
#SOLR_SERVER=5
#SOLR_SERVER=6
#SOLR_SERVER=7
SOLR_SERVER=latest

SOLR_CORE_NAME=devilbox
HOST_PORT_SOLR=8983
```

See also:

*.env file*

### 44.2.3 4. Start the Devilbox

The final step is to start the Devilbox with Solr.

Let's assume you want to start php, httpd, bind, solr.

```
host> docker-compose up -d php httpd bind solr
```

See also:

*Start the Devilbox*

## 44.3 TL;DR

For the lazy readers, here are all commands required to get you started. Simply copy and paste the following block into your terminal from the root of your Devilbox git directory:

```
# Copy compose-override.yml into place
cp compose/docker-compose.override.yml-solr docker-compose.override.yml

# Create .env variable
echo "# Solr version to choose"          >> .env
echo "#SOLR_SERVER=5"                    >> .env
echo "#SOLR_SERVER=6"                    >> .env
echo "#SOLR_SERVER=7"                    >> .env
echo "SOLR_SERVER=latest"                 >> .env
echo "SOLR_CORE_NAME=devilbox"            >> .env
echo "HOST_PORT_SOLR=8983"                >> .env

# Start container
docker-compose up -d php httpd bind solr
```



---

## Shared Devilbox server in LAN

---

Devilbox as a shared **development**, **staging** or **CI** server is setup in a similar way as you would do locally. The only three important parts to take care of are:

1. Project access to deploy/update code
2. Handle DNS entries
3. Share Devilbox CA

### Table of Contents

- *Prerequisites*
- *Project access*
  - *SSH*
    - \* *Copy via sftp*
    - \* *Manually git pull/checkout*
    - \* *Automated git pull/checkout*
  - *Samba*
- *Handle DNS records*
  - *Use a real domain*
  - *Handle DNS records in your own DNS server*
  - *Run a second instance of the Devilbox DNS server*
    - \* *Manual DNS settings*
    - \* *DHCP distributed*
      - *Self-managed DHCP server*

- *DSL box / LAN or WIFI router*
- *Add hosts entries for every project*
- *Share Devilbox CA*

## 45.1 Prerequisites

This walk-through will use the following example values:

LAN / Network	Devilbox server	TLD_SUFFIX	LOCAL_LISTEN_ADDR
192.168.0.0/24	192.168.0.12	loc	192.168.0.12 or empty

See also:

- *TLD\_SUFFIX*
- *LOCAL\_LISTEN\_ADDR*

## 45.2 Project access

### 45.2.1 SSH

Enable and start an SSH server and give access to whatever system or user requires it. This can be done directly on the host system or via various other Docker container that offer ssh server.

#### Copy via sftp

If your SSH server is setup, users can use their sftp clients to deploy code updates. This however is not encouraged and you should use git or any other version control system.

#### Manually git pull/checkout

When using git, users can directly ssh into the shared Devilbox server and `git pull` or `git checkout <branch>` on their projects.

#### Automated git pull/checkout

In case you are using a staging or CI server, use Jenkins jobs or other automation tools (e.g. Ansible) to auto-deploy via SSH.

### 45.2.2 Samba

For a shared development server, you could also setup Samba network shares for each projects and have users deploy their code via Samba.

## 45.3 Handle DNS records

There are multiple ways of having DNS records available accross the LAN.

Before you read on, have a quick look on the decision Matrix to find the best method for your use-case.

Method	Sub-method	Outcome
Real domain		All network devices will have Auto DNS
Own DNS server		All network devices will have Auto DNS
Devilbox DNS server	Manual	Every network device must configure its DNS settings
	DHCP distributed	All network devices will have Auto DNS
Hosts entry		Every network device must manually set hosts entries for each project. (Does not work for phones)

---

**Important:** When using a shared Devilbox server and another Devilbox setup on your local computer, ensure that you are using different *TLD\_SUFFIX* in order to not confuse DNS records.

---

### 45.3.1 Use a real domain

*(This will allow all devices on the network to have Auto-DNS)*

If you own a real domain, such as `my-company.com`, you can create a wildcard DNS record for a subdomain, such as `*.dev.my-company.com` which must point to `192.168.0.12..` This should be done in your hosting provider's DNS configuration pannel.

You must then also change the `TLD_SUFFIX` to that subdomain.

Listing 1: `.env`

```
TLD_SUFFIX=dev.my-company.com
```

### 45.3.2 Handle DNS records in your own DNS server

*(This will allow all devices on the network to have Auto-DNS)*

If your LAN already provides its own customizable DNS server, you can setup a new wildcard DNS zone for `*.loc` which points to `192.168.0.12.`

### 45.3.3 Run a second instance of the Devilbox DNS server

If the above two methods for automated DNS records don't apply to you, you will need to run a second stand-alone Docker container of the Devilbox DNS server.

Run this container permantently on the shared Devilbox server with the following command:

```
host> docker run -d \
    --restart unless-stopped \
    -p 53:53/tcp \
    -p 53:53/udp \
```

(continues on next page)

(continued from previous page)

```
-e WILDCARD_DNS='loc=192.168.0.12' \
-t cytopia/bind
```

**See also:**

<https://github.com/cytopia/docker-bind>

Now there are two ways to consume the DNS records on your local machine:

1. Manual
2. DHCP distributed

## Manual DNS settings

*(Each device on the network needs to manually set the DNS server)*

When using this approach, you have to manually add the DNS server (IP: 192.168.0.12) to your host operating system.

---

**Important:** Keep in mind that you have to do this for every machine within the network which wants to access the shared Devilbox server.

---

**See also:**

- *Add custom DNS server on Linux*
- *Add custom DNS server on MacOS*
- *Add custom DNS server on Windows*
- *Add custom DNS server on Android*
- *Add custom DNS server on iPhone*

## DHCP distributed

*(This will allow all devices on the network to have Auto-DNS)*

This is the automated and more pain-free approach, as all devices within the network will be able to access projects on the shared Devilbox server.

## Self-managed DHCP server

If you run your own DHCP server within a network, you probably know how to add other DNS servers. The only thing you should keep in mind is, that the Devilbox DNS server should be the first in the list.

## DSL box / LAN or WIFI router

Most **SOHO** networks probably use some vendor router which has a web interface. Generally speaking, you need to find the DNS/DHCP settings in its web interface and add the Devilbox DNS server as the first in the list (192.168.0.12).

**See also:**

- [Change DNS server in Fritzbox](#)

#### 45.3.4 Add hosts entries for every project

*(Each device on the network needs to manually set the hosts entries for every single project)*

As you also do for the Devilbox locally when not using Auto-DNS, you can do as well for remote computer. Just edit your local hosts file and add one DNS entry for every project on the shared Devilbox server.

Keep in mind that this time you will have to use `192.168.0.12` instead of `127.0.0.1`.

**See also:**

- [Add project hosts entry on Linux](#)
- [Add project hosts entry on MacOS](#)
- [Add project hosts entry on Windows](#)

### 45.4 Share Devilbox CA

The last step to also have valid HTTPS connections on your shared Devilbox server is to copy the CA onto your local machine and import it into your browser or system.

**See also:**

*[Setup valid HTTPS](#)*



---

## Use external databases

---

### Table of Contents

- *Why*
- *Database on host os*
- *Database on network*
- *Database on internet*

## 46.1 Why

Some people or companies might have concerns with dockerized databases and rather rely on good old host-based database setups. There could already be a database cluster in your network or you rather want to use AWS RDS or other cloud-based solutions.

There are many reasons for having an external database.

## 46.2 Database on host os

---

**Note:** If the local database is listening on an IP address that is reachable over your current LAN, you can directly skip to: *Database on network*

---

In order to use an already existing database that is running on the host os, you need to make sure the following is met:

1. Be able to connect to the host os from inside the container

**See also:**

*Connect to host OS*

2. Configure your application to use the IP/CNAME of the host os
3. When starting the Devilbox, explicitly specify the service to use and exclude the databases:

```
# Explicitly specify services to start (otherwise all will start)
# Omit the database
host> docker-compose up -d php httpd bind redis
```

**See also:**

*Start the Devilbox*

## 46.3 Database on network

In order to use an already existing database that is running on the network, you need to make sure the following is met:

1. Configure your application to use the IP/CNAME of the database host
2. When starting the Devilbox, explicitly specify the service to use and exclude the databases:

```
# Explicitly specify services to start (otherwise all will start)
# Omit the database
host> docker-compose up -d php httpd bind redis
```

**See also:**

*Start the Devilbox*

## 46.4 Database on internet

In order to use an already existing database that is running on the network, you need to make sure the following is met:

1. Configure your application to use the IP/CNAME of the database host
2. When starting the Devilbox, explicitly specify the service to use and exclude the databases:

```
# Explicitly specify services to start (otherwise all will start)
# Omit the database
host> docker-compose up -d php httpd bind redis
```

**See also:**

*Start the Devilbox*



---

### Checkout different Devilbox 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 a 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.



---

### Remove stopped container

---

#### 48.1 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.

#### 48.2 How to do it?

```
host> docker-compose stop
host> docker-compose rm
```

#### 48.3 When to do it?

Whenever path values inside the `.env` file change.



---

### 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*
  - *Keep vhost-gen templates 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*

## 49.1 Update git repository

### 49.1.1 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
```

### 49.1.2 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
```

### 49.1.3 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
```

### 49.1.4 Keep `.env` file in sync

---

**Important:** 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
```

### 49.1.5 Keep `vhost-gen` templates in sync

---

**Important:** Whenever you check out a different version, make sure that the vhost-gen templates that have been copied to any virtual hosts are up-to-date with the templates provided in `cfg/vhost-gen/`.

---

### 49.1.6 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*

## 49.2 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.

---

### 49.2.1 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.

### 49.2.2 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.

### 49.2.3 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
```

## 49.3 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. Ensure all of your custom applied `vhost-gen` templates are in sync with the default templates

## 49.4 Checklist Docker images

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



---

### Remove the Devilbox

---

If you want to completely remove the Devilbox follow this tutorial.

#### Table of Contents

- *Backups*
  - *Dump databases*
  - *Shutdown the Devilbox*
  - *Backup configuration files*
  - *Backup data and dumps*
- *Remove the Devilbox*
  - *Remove Devilbox containers*
  - *Remove Devilbox network*
  - *Remove Devilbox git directory*
- *Revert your system changes*
  - *AutoDNS*
  - *Manual DNS entries*
  - *Remove Devilbox CA from your browser*

### 50.1 Backups

Before deleting the Devilbox git directory, ask yourself if you want to make backups of all customizations you have done so far as well as of all data that may be present within that directory.

### 50.1.1 Dump databases

Before shutting down the Devilbox, do a final backup of all of your databases:

See also:

- *Backup and restore MySQL*
- *Backup and restore PostgreSQL*
- *Backup and restore MongoDB*

Dumps will end up in `backups/`.

### 50.1.2 Shutdown the Devilbox

Before attempting to backup any file system data, make sure the Devilbox is properly shutdown.

```
host> docker-compose stop
```

### 50.1.3 Backup configuration files

You should now backup the following configuration files:

- Backup your customized `.env` file
- Backup your customized `.docker-compose.override.yml` file
- Backup your customized bash configuration from `bash/`
- Backup all custom service configurations from `cfg/`
- Backup the Devilbox root certificate from `ca/`

### 50.1.4 Backup data and dumps

You should now backup the following data:

- Backup any backups created in `backups/`
- Backup any project or Docker data from `data/`

## 50.2 Remove the Devilbox

If you have followed the backup routine, you can continue deleting all created components.

### 50.2.1 Remove Devilbox containers

Navigate to the Devilbox git directory and remove all Devilbox container:

```
host> docker-compose rm -f
```

## 50.2.2 Remove Devilbox network

1. List all existing Docker networks via

```
host> docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
0069843ff0c3	bridge	bridge	local
...			
9c8d4a84cf2d	devilbox_app_net	bridge	local

2. Find the NETWORK ID of the Devilbox network and delete it:

```
host> docker network rm 9c8d4a84cf2d
```

## 50.2.3 Remove Devilbox git directory

You can simply delete the whole Devilbox git directory

# 50.3 Revert your system changes

## 50.3.1 AutoDNS

Revert any changes you have done for Auto-DNS to work.

**See also:**

*Setup Auto DNS*

## 50.3.2 Manual DNS entries

Revert any changes you have done in `/etc/hosts` (or `C:\Windows\System32\drivers\etc` for Windows)

**See also:**

- *Add project hosts entry on MacOS*
- *Add project hosts entry on Windows*

## 50.3.3 Remove Devilbox CA from your browser

Remove the Devilbox CA from your browser

**See also:**

- *Setup valid HTTPS*



---

## 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*

## 51.1 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.

### 51.1.1 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

```

### 51.1.2 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:



### 51.1.3 phpMyAdmin

If you do not like to use the command line for backups, you can use . It comes bundled with the devilbox intranet.

### 51.1.4 Adminer

If you do not like to use the command line for backups, you can use . It comes bundled with the devilbox intranet.

## 51.2 Restore

### 51.2.1 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
```

(continues on next page)

(continued from previous page)

---

### 51.2.2 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.

### 51.2.3 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.

---

### 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*
  - *phpPgAdmin*
  - *Adminer*
- *Restore*
  - *psql*
    - \* *\*.sql file*
    - \* *\*.sql.gz file*
    - \* *\*.sql.tar.gz file*
  - *phpPgAdmin*
  - *Adminer*

## 52.1 Backup

### 52.1.1 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
```

### 52.1.2 phpPgAdmin

If you do not like to use the command line for backups, you can use `.`. It comes bundled with the devilbox intranet.

### 52.1.3 Adminer

If you do not like to use the command line for backups, you can use `.`. It comes bundled with the devilbox intranet.

## 52.2 Restore

### 52.2.1 psql

In order to restore or import PostgreSQL databases on the command line, you need to use `.`. 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
```

## 52.2.2 phpPgAdmin

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.

## 52.2.3 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.



---

### 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*

## 53.1 Backup

### 53.1.1 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

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

## 53.2 Restore

### 53.2.1 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
```



---

### .env file

---

All docker-compose configuration is done inside the `.env` file which simply defines key-value pairs evaluated by docker-compose.yml.

If this file does not exist at the root of your Devilbox git directory, then copy `env-example` to `.env` to initially create it with sane defaults.

**See also:**

what is the 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*
  - *MOUNT\_OPTIONS*
  - *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*
    - \* *PHP\_MODULES\_ENABLE*

- \* *PHP\_MODULES\_DISABLE*
- \* *Custom variables*
- *Web server*
  - \* *HTTPD\_DOCROOT\_DIR*
  - \* *HTTPD\_TEMPLATE\_DIR*
  - \* *HTTPD\_TIMEOUT\_TO\_PHP\_FPM*
- *MySQL*
  - \* *MYSQL\_ROOT\_PASSWORD*
  - \* *MYSQL\_GENERAL\_LOG*
- *PostgreSQL*
  - \* *PGSQL\_ROOT\_USER*
  - \* *PGSQL\_ROOT\_PASSWORD*
- *Redis*
  - \* *REDIS\_ARGS*
    - *Example: Adding password protection*
    - *Example: Increasing verbosity*
    - *Example: Combining options*
- *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*

## 54.1 Core settings

### 54.1.1 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

### 54.1.2 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 sent 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.

### 54.1.3 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	.

### 54.1.4 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.

### 54.1.5 TLD\_SUFFIX

This variable controls all of your projects domain suffix.

Name	Allowed values	Default value
TLD_SUFFIX	alpha-num string	loc

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	<code>http://my-test.loc</code>
example	loc	<code>http://example.loc</code>
www.test	loc	<code>http://www.test.loc</code>
my-test	local	<code>http://my-test.local</code>
example	local	<code>http://example.local</code>
www.test	local	<code>http://www.test.local</code>

**Warning:** Do not use `dev` as a domain suffix (I know, it's tempting). It has been registered by and they advertise the which makes your browser redirect every http request to https.

**See also:**

**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:** and

**Warning:** Do not use official domain endings such as `.com`, `.org`, `.net`, etc. If you do, all name resolutions to any `.com` address (e.g.: `google.com`) will be resolved to the Devilbox's PHP server IP address.

The bundled DNS server does a catch-all on the given TLD\_SUFFIX and resolves everything below it to the PHP container.

### 54.1.6 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 to add external links.

**See also:**

*Connect to external hosts*

### 54.1.7 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.

### 54.1.8 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.

### 54.1.9 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:

---

**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).

---



## 54.2 Intranet settings

### 54.2.1 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

### 54.2.2 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:

*Setup valid HTTPS*

### 54.2.3 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`.

---

### 54.2.4 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
<code>DEVILBOX_UI_PASSWORD</code>	any string	password

### 54.2.5 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
<code>DEVILBOX_UI_ENABLE</code>	0 or 1	1

## 54.3 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.

---

### 54.3.1 PHP\_SERVER

This variable chooses your desired PHP-FPM version to be started.

Name	Allowed values	Default value
<code>PHP_SERVER</code>	<code>php-fpm-5.2</code> <code>php-fpm-5.3</code> <code>php-fpm-5.4</code> <code>php-fpm-5.5</code> <code>php-fpm-5.6</code> <code>php-fpm-7.0</code> <code>php-fpm-7.1</code> <code>php-fpm-7.2</code> <code>php-fpm-7.3</code> <code>php-fpm-7.4</code>	<code>php-fpm-7.2</code>

---

**Important:** **PHP 5.2** is available to use, but it is not officially supported. The Devilbox intranet does not work with this version as PHP 5.2 does not support namespaces. Furthermore PHP 5.2 does only work with Apache 2.4, Nginx stable and Nginx mainline. It does not work with Apache 2.2. **Use at your own risk.**

---

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-5.2
#PHP_SERVER=php-fpm-5.3
#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=php-fpm-7.4
```

### 54.3.2 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
```

### 54.3.3 MYSQL\_SERVER

This variable chooses 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
```

(continues on next page)

(continued from previous page)

```
#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
```

### 54.3.4 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.

---

### 54.3.5 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.

### 54.3.6 MEMCD\_SERVER

This variable chooses 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.

### 54.3.7 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.

---

## 54.4 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.

### 54.4.1 MOUNT\_OPTIONS

This variable allows you to add custom mount options/flags to all mounted directories. Initially only `rw` or `ro` are applied to mount points, you can however extend this before starting up the Devilbox.

Name	Allowed values	Default value
MOUNT_OPTIONS	valid mount option	empty

If you are on Linux with SELinux enabled, you will want to set this value to `,z` to modify SELinux labels in order to share mounts among multiple container.

See also:

- 
-

•

**Important:** When adding custom mount options, ensure to start with a leading `,`, as those options are prepended to already existing options.

```
MOUNT_OPTIONS=,z
MOUNT_OPTIONS=,cached
```

### 54.4.2 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`.

### 54.4.3 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`.

### 54.4.4 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`.

### 54.4.5 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`.

## 54.5 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 and the Devilbox* 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.

## 54.5.1 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

## 54.5.2 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.

Name	Allowed values	Default value
HOST_PORT_HTTPD_SSL	1 - 65535	443

## 54.5.3 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

#### 54.5.4 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

#### 54.5.5 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	5432

#### 54.5.6 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

#### 54.5.7 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

#### 54.5.8 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	1053

**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: *Setup Auto DNS*.

## 54.6 Container settings

### 54.6.1 PHP

#### PHP\_MODULES\_ENABLE

Enable any non-standard PHP modules in a comma separated list.

Name	Allowed values	Default value
PHP_MODULES_ENABLE	comma separated list of module names	empty

---

**Note:** Currently only `ioncube` is available to enable.

---

Example:

Listing 9: `.env`

```
# Enable ionCube
PHP_MODULES_ENABLE=ioncube
```

#### PHP\_MODULES\_DISABLE

Disable any PHP modules in a comma separated list.

Name	Allowed values	Default value
PHP_MODULES_DISABLE	comma separated list of module names	<code>blackfire,oci8,PDO_OCI,pdo_sqlsrv,sqlsrv,rdkafka,swoole</code>

Example:

Listing 10: `.env`

```
# Disable Xdebug, Imagick and Swoole
PHP_MODULES_DISABLE=xdebug,imagick,swoole
```

#### 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 11: `.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 12: `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:

*Add custom environment variables*

## 54.6.2 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 built up the final location of a virtual hosts document root.

Name	Allowed values	Default value
<code>HTTPD_DOCROOT_DIR</code>	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 `HTTDP_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 `cfg/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 cfg/vhost-gen/*.yaml 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/

# 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.yaml
-rw-r--r-- 1 cytopia cytopia 87 Mar 12 23:05 apache24.yaml
-rw-r--r-- 1 cytopia cytopia 87 Mar 12 23:05 nginx.yaml
```

The three files `apache22.yaml`, `apache24.yaml` and `nginx.yaml` 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 `.`. 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:

- `vhost-gen`: [Virtual host templates](#)

- vhost-gen: *Customize all virtual hosts globally*
- vhost-gen: *Customize specific virtual host*
- vhost-gen: *Example: add sub domains*

## HTTPD\_TIMEOUT\_TO\_PHP\_FPM

This variable specifies after how many seconds the webserver should quit an unanswered connection to PHP-FPM.

Ensure that this value is higher than PHP's `max_execution_time`, otherwise the PHP script could still run and the webserver will simply drop the connection before getting an answer by PHP.

If `HTTPD_TIMEOUT_TO_PHP_FPM` is smaller than `max_execution_time` and a script runs longer than `max_execution_time`, you will get a: 504 Gateway timeout in the browser.

If `HTTPD_TIMEOUT_TO_PHP_FPM` is greater than `max_execution_time` and a script runs longer than `max_execution_time`, you will get a proper PHP error message in the browser.

Name	Allowed values	Default value
<code>HTTPD_TIMEOUT_TO_PHP_FPM</code>	positive integer	180

## 54.6.3 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
<code>MYSQL_ROOT_PASSWORD</code>	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
<code>MYSQL_GENERAL_LOG</code>	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.”

—



## 54.6.4 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.

## 54.6.5 Redis

### REDIS\_ARGS

This option lets you add extra startup parameters to Redis. This could include adding a password protection to Redis or increasing its verbosity.

Name	Allowed values	Default value
REDIS_ARGS	valid redis-server startup parameter	empty

#### Example: Adding password protection

```
REDIS_ARGS=--requirepass my-redis-root-password
```

**Important:** Do not quote the password and do not use spaces inside the password.

### Example: Increasing verbosity

```
REDIS_ARGS=--loglevel verbose
```

### Example: Combining options

```
REDIS_ARGS=--loglevel verbose --requirepass my-redis-root-password
```

## 54.6.6 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\_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\_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\_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\_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:**

---

### 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:

## 56.1 Create docker-compose.override.yml

### 56.1.1 Copy example file

```
host> cd path/to/devilbox
host> cp docker-compose.override.yml-example docker-compose.override.yml
```

### 56.1.2 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.

---

## 56.2 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\*](#)



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:

- vhost-gen: *Virtual host templates*
  - vhost-gen: *Customize all virtual hosts globally*
  - vhost-gen: *Customize specific virtual host*
  - vhost-gen: *Example: add sub domains*
- 

### Table of Contents

- *General*
- *Examples*
  - *Adjust KeepAlive settings for Apache 2.2*
  - *Limit HTTP headers and GET size for Apache 2.4*

## 57.1 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.

## 57.2 Examples

### 57.2.1 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.

---

### 57.2.2 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
```

(continues on next page)

(continued from previous page)

```
# 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:

- vhost-gen: *Virtual host templates*
  - vhost-gen: *Customize all virtual hosts globally*
  - vhost-gen: *Customize specific virtual host*
  - vhost-gen: *Example: add sub domains*
- 

### Table of Contents

- *General*
  - *Examples*
    - *Adjust KeepAlive settings for Nginx stable*
    - *Adjust timeout settings for Nginx mainline*

## 58.1 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.

## 58.2 Examples

### 58.2.1 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.

---

### 58.2.2 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
```

(continues on next page)

(continued from previous page)

```
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*

## 59.1 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-5.2/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-ini-5.3/
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/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-ini-7.3/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-ini-7.4/
```

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 contains two example files: `devilbox-php.ini-default` and `devilbox-php.ini-xdebug`.

#### **devilbox-php.ini-default**

This file holds the exact settings that are currently in place by each PHP-FPM container. Copy it (do not simply rename it) to a different file ending by `.ini` and start adjusting it.

#### **devilbox-php.ini-xdebug**

This file holds some sane example configuration to get you started with Xdebug. Copy it (do not simply rename it) to a different file ending by `.ini` and start adjusting it.

---

**Important:** For Xdebug to work, there are other changes requires as well: [Configure PHP Xdebug](#)

---

#### **How to apply the settings**

In order for the changes to be applied, you will have to restart the Devilbox.

## **59.2 Examples**

### **59.2.1 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.

### **59.2.2 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
```

(continues on next page)

(continued from previous page)

```
# 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.



## CHAPTER 60

---

### php-fpm.conf

---

`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*

### 60.1 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-5.2/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-fpm-5.3/
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/
```

(continues on next page)

(continued from previous page)

```
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-fpm-7.0/
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/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-fpm-7.3/
drwxr-xr-x  2 cytopia cytopia 4096 Mar  5 21:53 php-fpm-7.4/
```

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 contains three example file: `devilbox-fpm.conf-default`, `devilbox-fpm.conf-pm_dynamic` and `devilbox-fpm.conf-pm_ondemand`.

#### **devilbox-fpm.conf-default**

This file holds the exact settings that are currently in place by each PHP-FPM container. Copy it (do not simply rename it) to a different file ending by `.conf` and start adjusting it.

#### **devilbox-fpm.conf-pm\_dynamic**

This file holds some sane example configuration to switch PHP-FPM scheduler to `dynamic` (The default is `ondemand`). Copy it (do not simply rename it) to a different file ending by `.conf` and start adjusting it.

#### **devilbox-fpm.conf-pm\_ondemand**

This file holds the current default values for the PHP-FPM scheduler which is using `ondemand`. Copy it (do not simply rename it) to a different file ending by `.conf` and start adjusting it.

#### **How to apply the settings**

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>

## 60.2 Examples

### 60.2.1 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.

## 60.2.2 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
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.

## 60.2.3 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 and the Devilbox* 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*

## 61.1 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.

## 61.2 Examples

### 61.2.1 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.

### 61.2.2 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 PHP container*

**Table of Contents**

- *Directory mapping*
- *Examples*
  - *Custom aliases*
  - *Custom vim configuration*

## 62.1 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>

## 62.2 Examples

### 62.2.1 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'
```

### 62.2.2 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.





## Setup CakePHP

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

After completing the below listed steps, you will have a working CakePHP setup ready to be served via http and https.

**See also:**

**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*

### 63.1 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	<a href="http://my-cake.loc">http://my-cake.loc</a> <a href="https://my-cake.loc">https://my-cake.loc</a>

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via `HOST_PATH_HTTPD_DATADIR`.
- 

## 63.2 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

### 63.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

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

### 63.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. (`<vhost dir>.TLD_SUFFIX` will be the final URL ).

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

**See also:**

*TLD\_SUFFIX*

### 63.2.3 3. Install CakePHP

Navigate into your newly created vhost directory and install CakePHP with `composer`.

```
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
```

How does the directory structure look after installation:

```
devilbox@php-7.0.20 in /shared/httpd/my-cake $ tree -L 1
.
└─ cakephp

1 directory, 0 files
```

### 63.2.4 4. Symlink webroot

Symlinking the actual webroot directory to `htdocs` is important. The web server expects every project's document root to be in `<vhost dir>/htdocs/`. This is the path where it will serve the files. This is also the path where your frameworks entrypoint (usually `index.php`) should be found.

Some frameworks however provide its actual content in nested directories of unknown levels. This would be impossible to figure out by the web server, so you manually have to symlink it back to its expected path.

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

How does the directory structure look after symlinking:

```
devilbox@php-7.0.20 in /shared/httpd/my-cake $ tree -L 1
.
├─ cakephp
└─ htdocs -> cakephp/webroot

2 directories, 0 files
```

As you can see from the above directory structure, `htdocs` is available in its expected path and points to the frameworks entrypoint.

### 63.2.5 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;'
```

### 63.2.6 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,
    ],
],
?>
```

### 63.2.7 7. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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:

- *Add project hosts entry on MacOS*
- *Add project hosts entry on Windows*
- *Setup Auto DNS*

### 63.2.8 8. Open your browser

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

#### See also:

*Setup valid HTTPS*

---

## Setup CodeIgniter

---

This example will use `wget` to install CodeIgniter from within the Devilbox PHP container.

After completing the below listed steps, you will have a working CodeIgniter setup ready to be served via http and https.

**See also:**

### Table of Contents

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

## 64.1 Overview

The following configuration will be used:

Project name	VirtualHost directory	Database	TLD_SUFFIX	Project URL
my-ci	/shared/httpd/my-ci	my_ci	loc	<a href="http://my-ci.loc">http://my-ci.loc</a> <a href="https://my-ci.loc">https://my-ci.loc</a>

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via [\*HOST\\_PATH\\_HTTPD\\_DATADIR\*](#).
- 

## 64.2 Walk through

It will be ready in eight simple steps:

1. Enter the PHP container
2. Create a new VirtualHost directory
3. Download CodeIgniter
4. Symlink webroot directory
5. Add MySQL database
6. Configure database connection
7. Setup DNS record
8. Visit <http://my-ci.loc> in your browser

### 64.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

- [\*Enter the PHP container\*](#)
- [\*Work inside the PHP container\*](#)
- [\*Available tools\*](#)

### 64.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. (`<vhost dir>.TLD_SUFFIX` will be the final URL ).

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

See also:

*TLD\_SUFFIX*

### 64.2.3 3. Download CodeIgniter

Navigate into your newly created vhost directory and install CodeIgniter.

```
devilbox@php-7.0.20 in /shared/httpd $ cd my-ci
devilbox@php-7.0.20 in /shared/httpd/my-ci $ wget https://github.com/bcit-ci/
↳CodeIgniter/archive/3.1.8.tar.gz
devilbox@php-7.0.20 in /shared/httpd/my-ci $ tar xfvz 3.1.8.tar.gz
```

How does the directory structure look after installation:

```
devilbox@php-7.0.20 in /shared/httpd/my-ci $ tree -L 1
.
├── 3.1.8.tar.gz
└── CodeIgniter-3.1.8

1 directory, 1 file
```

### 64.2.4 4. Symlink webroot

Symlinking the actual webroot directory to `htdocs` is important. The web server expects every project's document root to be in `<vhost dir>/htdocs/`. This is the path where it will serve the files. This is also the path where your frameworks entrypoint (usually `index.php`) should be found.

Some frameworks however provide its actual content in nested directories of unknown levels. This would be impossible to figure out by the web server, so you manually have to symlink it back to its expected path.

```
devilbox@php-7.0.20 in /shared/httpd/my-ci $ ln -s CodeIgniter-3.1.8/ htdocs
```

How does the directory structure look after symlinking:

```
devilbox@php-7.0.20 in /shared/httpd/my-ci $ tree -L 1
.
├── 3.1.8.tar.gz
├── CodeIgniter-3.1.8
└── htdocs -> CodeIgniter-3.1.8

2 directories, 1 file
```

As you can see from the above directory structure, `htdocs` is available in its expected path and points to the frameworks entrypoint.

### 64.2.5 5. Add MySQL Database

```
devilbox@php-7.0.20 in /shared/httpd/my-ci $ mysql -u root -h 127.0.0.1 -p -e 'CREATE_
↳DATABASE my_ci;'
```

## 64.2.6 6. Configure database connection

```
devilbox@php-7.0.20 in /shared/httpd/my-ci $ vi htdocs/application/config/database.php
```

Listing 1: htdocs/application/config/database.php

```
<?php
$db['default'] = array(
    'dsn'      => '',
    'hostname' => '127.0.0.1',
    'username' => 'root',
    'password' => '',
    'database' => 'my_ci',
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => FALSE,
    'db_debug' => (ENVIRONMENT !== 'production'),
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt'  => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array(),
    'save_queries' => TRUE
);
```

## 64.2.7 7. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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-ci.loc
```

### See also:

- *Add project hosts entry on MacOS*
- *Add project hosts entry on Windows*
- *Setup Auto DNS*

## 64.2.8 8. Open your browser

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

### See also:

*Setup valid HTTPS*



---

## Setup CraftCMS

---

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

After completing the below listed steps, you will have a working CraftCMS setup ready to be served via http and https.

**See also:**

### Table of Contents

- *Overview*
- *Walk through*
  - *1. Enter the PHP container*
  - *2. Create new vhost directory*
  - *3. Install CraftCMS*
  - *4. Symlink webroot*
  - *5. Add MySQL Database*
  - *6. DNS record*
  - *7. Run setup wizard*
    - \* *7.1 Via command line tool*
    - \* *7.2 Via browser*
  - *8. Open your browser*

## 65.1 Overview

The following configuration will be used:

Project name	VirtualHost directory	Database	TLD_SUFFIX	Project URL
my-craft	/shared/httpd/my-craft	my_craft	loc	<a href="http://my-craft.loc">http://my-craft.loc</a> <a href="https://my-craft.loc">https://my-craft.loc</a>

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via [\*HOST\\_PATH\\_HTTPD\\_DATADIR\*](#).
- 

## 65.2 Walk through

It will be ready in eight simple steps:

1. Enter the PHP container
2. Create a new VirtualHost directory
3. Install CraftCMS via `composer`
4. Symlink webroot directory
5. Add MySQL database
6. Setup DNS record
7. Run setup wizard
8. Visit <http://my-craft.loc> in your browser

### 65.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

- [\*Enter the PHP container\*](#)
- [\*Work inside the PHP container\*](#)
- [\*Available tools\*](#)

### 65.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. (`<vhost dir>.TLD_SUFFIX` will be the final URL ).

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

See also:

[\*TLD\\_SUFFIX\*](#)

### 65.2.3 3. Install CraftCMS

Navigate into your newly created vhost directory and install CraftCMS with composer.

```
devilbox@php-7.0.20 in /shared/httpd $ cd my-craft
devilbox@php-7.0.20 in /shared/httpd/my-craft $ composer create-project craftcms/
↪craft craftcms
```

How does the directory structure look after installation:

```
devilbox@php-7.0.20 in /shared/httpd/my-craft $ tree -L 1
.
└─ craftcms

1 directory, 0 files
```

### 65.2.4 4. Symlink webroot

Symlinking the actual webroot directory to `htdocs` is important. The web server expects every project's document root to be in `<vhost dir>/htdocs/`. This is the path where it will serve the files. This is also the path where your frameworks entrypoint (usually `index.php`) should be found.

Some frameworks however provide its actual content in nested directories of unknown levels. This would be impossible to figure out by the web server, so you manually have to symlink it back to its expected path.

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

How does the directory structure look after symlinking:

```
devilbox@php-7.0.20 in /shared/httpd/my-craft $ tree -L 1
.
├─ craftcms
└─ htdocs -> craftcms/web

2 directories, 0 files
```

As you can see from the above directory structure, `htdocs` is available in its expected path and points to the frameworks entrypoint.

### 65.2.5 5. Add MySQL Database

```
devilbox@php-7.0.20 in /shared/httpd/my-craft $ mysql -u root -h 127.0.0.1 -p -e
↪'CREATE DATABASE my_craft CHARACTER SET utf8 COLLATE utf8_unicode_ci;'
```

### 65.2.6 6. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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-craft.loc
```

**See also:**

- *Add project hosts entry on MacOS*
- *Add project hosts entry on Windows*
- *Setup Auto DNS*

## 65.2.7 7. Run setup wizard

After everything is setup, you need to run the setup wizard. CraftCMS bundles a commandline tool that you can use.

### 7.1 Via command line tool

```
devilbox@php-7.0.20 in /shared/httpd/my-craft $ php craftcms/craft setup

Which database driver are you using? [mysql,pgsql,?]: mysql
Database server name or IP address: [localhost] 127.0.0.1
Database port: [3306]
Database username: [root]
Database password:
Database name: my_craft
Database table prefix:
Testing database credentials... success!
Saving database credentials to your .env file... done

Install Craft now? (yes|no) [yes]:

Username: [admin]
Email: admin@devilbox.org
Password:
Confirm:
Site name: craftcms
Site URL: [@web] my-craft.loc
Site language: [en-US]

...

*** installed Craft successfully (time: 14.660s)
```

### 7.2 Via browser

If you do not feel too comfortable on the command line, you can also run the setup wizard via the browser. See their official documentation for screenshots.

**See also:**

To open the setup wizard, visit: <http://my-craft.loc/admin/install> or <https://my-craft.loc/admin/install>

- Driver: MySQL
- Server: 127.0.0.1
- Port: 3306
- Username: root
- Password: your MySQL password
- Database Name: my\_craft
- Prefix: leave empty

---

**Important:** When choosing the Database server, use 127.0.0.1 as the hostname.

---

## 65.2.8 8. Open your browser

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

**See also:**

*Setup valid HTTPS*



## Setup Drupal

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

After completing the below listed steps, you will have a working Drupal setup ready to be served via http and https.

**See also:**

**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*

### 66.1 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	<a href="http://my-drupal.loc">http://my-drupal.loc</a> <a href="https://my-drupal.loc">https://my-drupal.loc</a>

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via `HOST_PATH_HTTPD_DATADIR`.
- 

## 66.2 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

### 66.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

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

### 66.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. ( `<vhost dir>.TLD_SUFFIX` will be the final URL ).

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

**See also:**

*TLD\_SUFFIX*



### 66.2.3 3. Install Drupal

Navigate into your newly created vhost directory and install Drupal with drush.

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

How does the directory structure look after installation:

```
devilbox@php-7.0.20 in /shared/httpd/my-drupal $ tree -L 1
.
└─ drupal-8.3.3

1 directory, 0 files
```

### 66.2.4 4. Symlink webroot

Symlinking the actual webroot directory to `htdocs` is important. The web server expects every project's document root to be in `<vhost dir>/htdocs/`. This is the path where it will serve the files. This is also the path where your frameworks entypoint (usually `index.php`) should be found.

Some frameworks however provide its actual content in nested directories of unknown levels. This would be impossible to figure out by the web server, so you manually have to symlink it back to its expected path.

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

How does the directory structure look after symlinking:

```
devilbox@php-7.0.20 in /shared/httpd/my-drupal $ tree -L 1
.
├─ drupal-8.3.3
└─ htdocs -> CodeIgniter-3.1.8

2 directories, 0 files
```

As you can see from the above directory structure, `htdocs` is available in its expected path and points to the frameworks entypoint.

### 66.2.5 5. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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:

- [Add project hosts entry on MacOS](#)
- [Add project hosts entry on Windows](#)
- [Setup Auto DNS](#)

### 66.2.6 6. Open your browser

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

---

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

---

**See also:**

*Setup valid HTTPS*

# CHAPTER 67

## Setup Joomla

This example will use `wget` to install Joomla from within the Devilbox PHP container.

After completing the below listed steps, you will have a working Joomla setup ready to be served via http and https.

**See also:**

**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*

## 67.1 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	<a href="http://my-joomla.loc">http://my-joomla.loc</a> <a href="https://my-joomla.loc">https://my-joomla.loc</a>

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via `HOST_PATH_HTTPD_DATADIR`.
- 

## 67.2 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

### 67.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

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

### 67.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. ( `<vhost dir>.TLD_SUFFIX` will be the final URL ).

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

**See also:**

*TLD\_SUFFIX*

### 67.2.3 3. Download and extract Joomla

Navigate into your newly created vhost directory and install 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/my-joomla $ mkdir joomla
devilbox@php-7.0.20 in /shared/httpd/my-joomla $ tar xvfz joomla.tar.gz -C joomla/
```

How does the directory structure look after installation:

```
devilbox@php-7.0.20 in /shared/httpd/my-joomla $ tree -L 1
.
├── joomla.tar.gz
└── joomla

1 directory, 1 file
```

### 67.2.4 4. Symlink webroot

Symlinking the actual webroot directory to `htdocs` is important. The web server expects every project's document root to be in `<vhost dir>/htdocs/`. This is the path where it will serve the files. This is also the path where your frameworks entrypoint (usually `index.php`) should be found.

Some frameworks however provide its actual content in nested directories of unknown levels. This would be impossible to figure out by the web server, so you manually have to symlink it back to its expected path.

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

How does the directory structure look after symlinking it:

```
devilbox@php-7.0.20 in /shared/httpd/my-joomla $ tree -L 1
.
├── joomla.tar.gz
├── joomla
└── htdocs -> joomla

2 directories, 1 file
```

As you can see from the above directory structure, `htdocs` is available in its expected path and points to the frameworks entrypoint.

### 67.2.5 5. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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:**

- *Add project hosts entry on MacOS*
- *Add project hosts entry on Windows*
- *Setup Auto DNS*

## 67.2.6 6. Open your browser

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

**See also:**

*Setup valid HTTPS*

Setup Laravel

---

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

After completing the below listed steps, you will have a working Laravel setup ready to be served via http and https.

**See also:**

**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*

## 68.1 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	<a href="http://my-laravel.loc">http://my-laravel.loc</a> <a href="https://my-laravel.loc">https://my-laravel.loc</a>

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via `HOST_PATH_HTTPD_DATADIR`.
- 

## 68.2 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

### 68.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

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

### 68.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. ( `<vhost dir>.TLD_SUFFIX` will be the final URL ).

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

**See also:**

*TLD\_SUFFIX*



### 68.2.3 3. Install Laravel

Navigate into your newly created vhost directory and install Laravel with `laravel cli`.

```
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
```

How does the directory structure look after installation:

```
devilbox@php-7.0.20 in /shared/httpd/my-laravel $ tree -L 1
.
└─ laravel-project

1 directory, 0 files
```

### 68.2.4 4. Symlink webroot

Symlinking the actual webroot directory to `htdocs` is important. The web server expects every project's document root to be in `<vhost dir>/htdocs/`. This is the path where it will serve the files. This is also the path where your frameworks entrypoint (usually `index.php`) should be found.

Some frameworks however provide its actual content in nested directories of unknown levels. This would be impossible to figure out by the web server, so you manually have to symlink it back to its expected path.

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

How does the directory structure look after symlinking:

```
devilbox@php-7.0.20 in /shared/httpd/my-laravel $ tree -L 1
.
├─ laravel-project
└─ htdocs -> laravel-project/public

2 directories, 0 files
```

As you can see from the above directory structure, `htdocs` is available in its expected path and points to the frameworks entrypoint.

### 68.2.5 5. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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
```

This will ensure that your host operating system's browser will direct any calls on `http://my-laravel.loc` or `https://my-laravel.loc` to the Devilbox which is listening on `127.0.0.1`.

See also:

- [Add project hosts entry on MacOS](#)

- *Add project hosts entry on Windows*
- *Setup Auto DNS*

### 68.2.6 6. Open your browser

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

**See also:**

*Setup valid HTTPS*

---

### Setup Magento 2

---

This example will use `git` and `composer` to install Magento 2 from within the Devilbox PHP container.

After completing the below listed steps, you will have a working Magento 2 setup ready to be served via `http` and `https`.

**See also:**

#### Table of Contents

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

### 69.1 Overview

The following configuration will be used:

Project name	VirtualHost directory	Database	TLD_SUFFIX	Project URL
my-magento	/shared/httpd/my-magento	my_magento	loc	<a href="http://my-magento.loc">http://my-magento.loc</a> <a href="https://my-magento.loc">https://my-magento.loc</a>

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via [\*HOST\\_PATH\\_HTTPD\\_DATADIR\*](#).
- 

## 69.2 Walk through

It will be ready in eight simple steps:

1. Enter the PHP container
2. Create a new VirtualHost directory
3. Install Magento 2 via `git` and `composer`
4. Symlink webroot directory
5. Add MySQL database
6. Setup DNS record
7. Visit <http://my-magento.loc> in your browser

### 69.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

- [\*Enter the PHP container\*](#)
- [\*Work inside the PHP container\*](#)
- [\*Available tools\*](#)

### 69.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. ( `<vhost dir>.TLD_SUFFIX` will be the final URL ).

```
devilbox@php-7.1.20 in /shared/httpd $ mkdir my-magento
```

See also:

*TLD\_SUFFIX*

### 69.2.3 3. Install Magento 2

Navigate into your newly created vhost directory and install Magento 2 with `git`.

```
devilbox@php-7.1.20 in /shared/httpd $ cd my-magento

# Download Magento 2 via git
devilbox@php-7.1.20 in /shared/httpd/my-magento $ git clone https://github.com/
↪magento/magento2

# Checkout the latest stable git tag
devilbox@php-7.1.20 in /shared/httpd/my-magento $ cd magento2
devilbox@php-7.1.20 in /shared/httpd/my-magento/magento2 $ git checkout 2.2.5

# Install dependencies with Composer
devilbox@php-7.1.20 in /shared/httpd/my-magento/magento2 $ composer install
```

How does the directory structure look after installation:

```
devilbox@php-7.1.20 in /shared/httpd/my-magento $ tree -L 1
.
└─ magento2

1 directory, 0 files
```

### 69.2.4 4. Symlink webroot

Symlinking the actual webroot directory to `htdocs` is important. The web server expects every project's document root to be in `<vhost dir>/htdocs/`. This is the path where it will serve the files. This is also the path where your frameworks entrypoint (usually `index.php`) should be found.

Some frameworks however provide its actual content in nested directories of unknown levels. This would be impossible to figure out by the web server, so you manually have to symlink it back to its expected path.

```
devilbox@php-7.1.20 in /shared/httpd/my-magento $ ln -s magento2/ htdocs
```

How does the directory structure look after symlinking:

```
devilbox@php-7.1.20 in /shared/httpd/my-magento $ tree -L 1
.
├─ magento2
└─ htdocs -> magento2

2 directories, 0 files
```

As you can see from the above directory structure, `htdocs` is available in its expected path and points to the frameworks entrypoint.

## 69.2.5 5. Add MySQL Database

```
devilbox@php-7.1.20 in /shared/httpd/my-magento $ mysql -u root -h 127.0.0.1 -p -e  
↪ 'CREATE DATABASE my_magento;'
```

## 69.2.6 7. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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-magento.loc
```

### See also:

- *Add project hosts entry on MacOS*
- *Add project hosts entry on Windows*
- *Setup Auto DNS*

## 69.2.7 8. Open your browser

All set now, you can visit <http://my-magento.loc> or <https://my-magento.loc> in your browser and follow the installation steps.

---

**Important:** Use `127.0.0.1` for the MySQL database hostname.

---

### See also:

*Setup valid HTTPS*

# CHAPTER 70

## Setup Phalcon

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

After completing the below listed steps, you will have a working Phalcon setup ready to be served via http and https.

**See also:**

### 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*
  - *7. Create custom vhost config file (Nginx Only)*

## 70.1 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	<a href="http://my-phalcon.loc">http://my-phalcon.loc</a> <a href="https://my-phalcon.loc">https://my-phalcon.loc</a>

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via `HOST_PATH_HTTPD_DATADIR`.
- 

## 70.2 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
7. (Nginx) Create custom vhost config file

### 70.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

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

### 70.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. ( `<vhost dir>.TLD_SUFFIX` will be the final URL ).

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

**See also:**

*[TLD\\_SUFFIX](#)*



### 70.2.3 3. Install Phalcon

Navigate into your newly created vhost directory and install Phalcon with `phalcon cli`.

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

How does the directory structure look after installation:

```
devilbox@php-7.0.20 in /shared/httpd/my-phalcon $ tree -L 1
.
└─ phalconphp
1 directory, 0 files
```

### 70.2.4 4. Symlink webroot

Symlinking the actual webroot directory to `htdocs` is important. The web server expects every project's document root to be in `<vhost dir>/htdocs/`. This is the path where it will serve the files. This is also the path where your frameworks entypoint (usually `index.php`) should be found.

Some frameworks however provide its actual content in nested directories of unknown levels. This would be impossible to figure out by the web server, so you manually have to symlink it back to its expected path.

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

How does the directory structure look after symlinking:

```
devilbox@php-7.0.20 in /shared/httpd/my-phalcon $ tree -L 1
.
├─ phalconphp
└─ htdocs -> phalconphp/public
2 directories, 0 files
```

As you can see from the above directory structure, `htdocs` is available in its expected path and points to the frameworks entypoint.

### 70.2.5 5. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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:

- [Add project hosts entry on MacOS](#)
- [Add project hosts entry on Windows](#)
- [Setup Auto DNS](#)

## 70.2.6 6. Open your browser

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

**See also:**

*Setup valid HTTPS*

## 70.2.7 7. Create custom vhost config file (Nginx Only)

By default routes will not work if using Nginx. To fix this, you will need to create a custom vhost configuration.

In your project folder, you will need to create a folder called `.devilbox` unless you changed `HTTPD_TEMPLATE_DIR` in your `.env`

Copy the default nginx config from `./cfg/vhost-gen/nginx.yml-example-vhost` to `./data/www/my-project/.devilbox/nginx.yml`

Carefully edit the `nginx.yml` file and change:

```
try_files $uri $uri/ /index.php$is_args$args; to try_files $uri $uri/ /index.  
php?_url=$uri&$args;
```

and

```
location ~ /\.php?$ { to location ~ [^/]\.php(/|$) {
```

save the file as `nginx.yml` and ensure not to use any tabs in the file or devilbox will not use the custom configuration. You can use `yamllint nginx.yml` whilst inside the Devilbox shell to check the file before restarting devilbox.

## Setup Photon CMS

This example will use `photon cli` to install Photon CMS from within the Devilbox PHP container.

After completing the below listed steps, you will have a working Laravel setup ready to be served via http and https.

**See also:**

**Table of Contents**

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

## 71.1 Overview

The following configuration will be used:

Project name	VirtualHost directory	Database	TLD_SUFFIX	Project URL
my-photon	/shared/httpd/my-photon	blog	loc	<a href="http://my-photon.loc">http://my-photon.loc</a> <a href="https://my-photon.loc">https://my-photon.loc</a>

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via `HOST_PATH_HTTPD_DATADIR`.
- 

## 71.2 Walk through

It will be ready in six simple steps:

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

### 71.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

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

### 71.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. ( `<vhost dir>.TLD_SUFFIX` will be the final URL ).

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

**See also:**

*[TLD\\_SUFFIX](#)*

### 71.2.3 3. Install Photon

Navigate into your newly created vhost directory and install Photom CMS with `photon cli`.

**Note:** During the installation you will be asked for the MySQL hostname, username and password. Ensure not to specify `localhost`, but instead use `127.0.0.1` for the hostname. Additionally, provide a pair of credentials that has permissions to create a database or create the database itself beforehand.

```
devilbox@php-7.0.20 in /shared/httpd $ cd my-photon
devilbox@php-7.0.20 in /shared/httpd/my-photon $ photon new blog
...What is your mysql hostname? [localhost] 127.0.0.1
...What is your mysql username? [root]root
...What is your mysql password? []
```

How does the directory structure look after installation:

```
devilbox@php-7.0.20 in /shared/httpd/my-photon $ tree -L 1
.
└─ blog

1 directory, 0 files
```

### 71.2.4 4. Symlink webroot

Symlinking the actual webroot directory to `htdocs` is important. The web server expects every project's document root to be in `<vhost dir>/htdocs/`. This is the path where it will serve the files. This is also the path where your frameworks entrypoint (usually `index.php`) should be found.

Some frameworks however provide its actual content in nested directories of unknown levels. This would be impossible to figure out by the web server, so you manually have to symlink it back to its expected path.

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

How does the directory structure look after symlinking:

```
devilbox@php-7.0.20 in /shared/httpd/my-photon $ tree -L 1
.
├─ blog
└─ htdocs -> blog/public

2 directories, 0 files
```

As you can see from the above directory structure, `htdocs` is available in its expected path and points to the frameworks entrypoint.

### 71.2.5 5. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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-photon.loc
```

**See also:**

- *Add project hosts entry on MacOS*
- *Add project hosts entry on Windows*
- *Setup Auto DNS*

## 71.2.6 6. Open your browser

Open your browser at <http://my-photon.loc> or <https://my-photon.loc>

**See also:**

*Setup valid HTTPS*

---

## Setup PrestaShop

---

This example will use `git` and `composer` to install PrestaShop from within the Devilbox PHP container.

After completing the below listed steps, you will have a working PrestaShop setup ready to be served via `http` and `https`.

**See also:**

### Table of Contents

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

## 72.1 Overview

The following configuration will be used:

Project name	VirtualHost directory	Database	TLD_SUFFIX	Project URL
my-presta	/shared/httpd/my-presta	my_presta	loc	<a href="http://my-presta.loc">http://my-presta.loc</a> <a href="https://my-presta.loc">https://my-presta.loc</a>

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via [\*HOST\\_PATH\\_HTTPD\\_DATADIR\*](#).
- 

## 72.2 Walk through

It will be ready in eight simple steps:

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

### 72.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

- [\*Enter the PHP container\*](#)
- [\*Work inside the PHP container\*](#)
- [\*Available tools\*](#)

### 72.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. (`<vhost dir>.TLD_SUFFIX` will be the final URL ).



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

**See also:**

*TLD\_SUFFIX*

### 72.2.3 3. Install PrestaShop

Navigate into your newly created vhost directory and install PrestaShop with git and composer.

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

# Download PrestaShop with git
devilbox@php-7.0.20 in /shared/httpd/my-presta $ git clone https://github.com/
↪PrestaShop/PrestaShop

# Checkout the latest stable git tag
devilbox@php-7.0.20 in /shared/httpd/my-presta $ cd PrestaShop
devilbox@php-7.0.20 in /shared/httpd/my-presta $ git checkout 1.7.4.2

# Install dependencies with Composer
devilbox@php-7.0.20 in /shared/httpd/my-presta $ composer install
```

How does the directory structure look after installation:

```
devilbox@php-7.0.20 in /shared/httpd/my-presta $ tree -L 1
.
├── PrestaShop

1 directory, 0 files
```

### 72.2.4 4. Symlink webroot

Symlinking the actual webroot directory to `htdocs` is important. The web server expects every project's document root to be in `<vhost dir>/htdocs/`. This is the path where it will serve the files. This is also the path where your frameworks entrypoint (usually `index.php`) should be found.

Some frameworks however provide its actual content in nested directories of unknown levels. This would be impossible to figure out by the web server, so you manually have to symlink it back to its expected path.

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

How does the directory structure look after symlinking:

```
devilbox@php-7.0.20 in /shared/httpd/my-presta $ tree -L 1
.
├── PrestaShop
├── htdocs -> PrestaShop

2 directories, 0 files
```

As you can see from the above directory structure, `htdocs` is available in its expected path and points to the frameworks entrypoint.

## 72.2.5 5. Add MySQL Database

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

## 72.2.6 6. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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-presta.loc
```

**See also:**

- *Add project hosts entry on MacOS*
- *Add project hosts entry on Windows*
- *Setup Auto DNS*

## 72.2.7 7. Open your browser

All set now, you can visit <http://my-presta.loc> or <https://my-presta.loc> in your browser and follow the installation steps.

**See also:**

*Setup valid HTTPS*

---

## Setup Shopware

---

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

After completing the below listed steps, you will have a working Laravel setup ready to be served via http and https.

**See also:**

- 
- 

**Table of Contents**

- *Overview*
- *Walk through*
  - *1. Enter the PHP container*
  - *2. Create new vhost directory*
  - *3. Download Shopware*
  - *4. Symlink webroot*
  - *5. Add MySQL Database*
  - *6. DNS record*
  - *7. Follow install steps in your browser*
- *Encountered problems*

## 73.1 Overview

The following configuration will be used:

Project name	VirtualHost directory	Database	TLD_SUFFIX	Project URL
my-sw	/shared/httpd/my-sw	my_sw	loc	<a href="http://my-sw.loc">http://my-sw.loc</a> <a href="https://my-sw.loc">https://my-sw.loc</a>

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via [\*HOST\\_PATH\\_HTTPD\\_DATADIR\*](#).
- 

## 73.2 Walk through

It will be ready in seven simple steps:

1. Enter the PHP container
2. Create a new VirtualHost directory
3. Download Shopware via `git`
4. Symlink webroot directory
5. Add MySQL database
6. Setup DNS record
7. Follow installation steps in <http://my-sw.loc> in your browser

### 73.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

- [\*Enter the PHP container\*](#)
- [\*Work inside the PHP container\*](#)
- [\*Available tools\*](#)

### 73.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. ( `<vhost dir> .TLD_SUFFIX` will be the final URL ).

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

**See also:**

[\*TLD\\_SUFFIX\*](#)

### 73.2.3 3. Download Shopware

Navigate into your newly created vhost directory and install Shopware with `git`.

```
devilbox@php-7.0.20 in /shared/httpd $ cd my-sw
devilbox@php-7.0.20 in /shared/httpd/my-sw $ git clone https://github.com/shopware/
↪shopware
```

How does the directory structure look after installation:

```
devilbox@php-7.0.20 in /shared/httpd/my-sw $ tree -L 1
.
├── shopware

1 directory, 0 files
```

### 73.2.4 4. Symlink webroot

Symlinking the actual webroot directory to `htdocs` is important. The web server expects every project's document root to be in `<vhost dir>/htdocs/`. This is the path where it will serve the files. This is also the path where your frameworks entypoint (usually `index.php`) should be found.

Some frameworks however provide its actual content in nested directories of unknown levels. This would be impossible to figure out by the web server, so you manually have to symlink it back to its expected path.

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

How does the directory structure look after symlinking:

```
devilbox@php-7.0.20 in /shared/httpd/my-sw $ tree -L 1
.
├── shopware
└── htdocs -> shopware

2 directories, 0 files
```

As you can see from the above directory structure, `htdocs` is available in its expected path and points to the frameworks entypoint.

### 73.2.5 5. Add MySQL Database

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

### 73.2.6 6. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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-sw.loc
```

**See also:**

- *Add project hosts entry on MacOS*
- *Add project hosts entry on Windows*
- *Setup Auto DNS*

### 73.2.7 7. Follow install steps in your browser

All set now, you can visit <http://my-sw.loc> or <https://my-sw.loc> in your browser and follow the installation steps as described in the :

---

**Important:** When setting up database connection use the following values:

- Database server: `127.0.0.1`
  - Database user: `root` (if you don't have a dedicated user already)
  - Database pass: by default the root password is empty
  - Database name: `my_sw`
- 

**See also:**

*Setup valid HTTPS*

## 73.3 Encountered problems

By the time of writing (2018-07-07) Shopware had loading issues with the combination of PHP 5.6 and Apache 2.4. Use any other combination.

**See also:**

<https://github.com/cytopia/devilbox/issues/300>

## Setup Symfony

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

After completing the below listed steps, you will have a working Symfony setup ready to be served via http and https.

**See also:**

**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*

## 74.1 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	<a href="http://my-symfony.loc">http://my-symfony.loc</a> <a href="https://my-symfony.loc">https://my-symfony.loc</a>

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via `HOST_PATH_HTTPD_DATADIR`.
- 

## 74.2 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

### 74.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

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

### 74.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. ( `<vhost dir>.TLD_SUFFIX` will be the final URL ).

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

**See also:**

*[TLD\\_SUFFIX](#)*



### 74.2.3 3. Install Symfony

Navigate into your newly created vhost directory and install Symfony with `symfony cli`.

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

How does the directory structure look after installation:

```
devilbox@php-7.0.20 in /shared/httpd/my-symfony $ tree -L 1
.
├── symfony

1 directory, 0 files
```

### 74.2.4 4. Symlink webroot

Symlinking the actual webroot directory to `htdocs` is important. The web server expects every project's document root to be in `<vhost dir>/htdocs/`. This is the path where it will serve the files. This is also the path where your frameworks entrypoint (usually `index.php`) should be found.

Some frameworks however provide its actual content in nested directories of unknown levels. This would be impossible to figure out by the web server, so you manually have to symlink it back to its expected path.

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

How does the directory structure look after symlinking:

```
devilbox@php-7.0.20 in /shared/httpd/my-sw $ tree -L 1
.
├── symfony
└── htdocs -> symfony/web

2 directories, 0 files
```

As you can see from the above directory structure, `htdocs` is available in its expected path and points to the frameworks entrypoint.

### 74.2.5 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
```

### 74.2.6 6. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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:**

- *Add project hosts entry on MacOS*
- *Add project hosts entry on Windows*
- *Setup Auto DNS*

## 74.2.7 7. Open your browser

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

**See also:**

*Setup valid HTTPS*

This example will use `composer` to install Typo3 from within the Devilbox PHP container.

After completing the below listed steps, you will have a working Laravel setup ready to be served via http and https.

**See also:**

**Table of Contents**

- *Overview*
- *Walk through*
  - *1. Enter the PHP container*
  - *2. Create new vhost directory*
  - *3. Install Typo3*
  - *4. Symlink webroot*
  - *5. DNS record*
  - *6. Create `FIRST_INSTALL` file*
  - *7. Open your browser*
  - *8. Step through guided web installation*

## 75.1 Overview

The following configuration will be used:

Project name	VirtualHost directory	Database	TLD_SUFFIX	Project URL
my-typo	/shared/httpd/my-typo	my_typo	loc	<a href="http://my-typo.loc">http://my-typo.loc</a> <a href="https://my-typo.loc">https://my-typo.loc</a>

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via [\*HOST\\_PATH\\_HTTPD\\_DATADIR\*](#).
- 

## 75.2 Walk through

It will be ready in eight simple steps:

1. Enter the PHP container
2. Create a new VirtualHost directory
3. Install Typo3 via `composer`
4. Symlink webroot directory
5. Setup DNS record
6. Create `FIRST_INSTALL` file
7. Open your browser
8. Step through guided web installation

### 75.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

- [\*Enter the PHP container\*](#)
- [\*Work inside the PHP container\*](#)
- [\*Available tools\*](#)

### 75.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. ( `<vhost dir>.TLD_SUFFIX` will be the final URL ).

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

**See also:**

[\*TLD\\_SUFFIX\*](#)

### 75.2.3 3. Install Typo3

Navigate into your newly created vhost directory and install Typo3 with `composer`.

```
devilbox@php-7.0.20 in /shared/httpd $ cd my-typo
devilbox@php-7.0.20 in /shared/httpd/my-typo $ composer create-project typo3/cms-base-
↪distribution typo3
```

How does the directory structure look after installation:

```
devilbox@php-7.0.20 in /shared/httpd/my-typo $ tree -L 1
.
└─ typo3

1 directory, 0 files
```

### 75.2.4 4. Symlink webroot

Symlinking the actual webroot directory to `htdocs` is important. The web server expects every project's document root to be in `<vhost dir>/htdocs/`. This is the path where it will serve the files. This is also the path where your frameworks entrypoint (usually `index.php`) should be found.

Some frameworks however provide its actual content in nested directories of unknown levels. This would be impossible to figure out by the web server, so you manually have to symlink it back to its expected path.

```
devilbox@php-7.0.20 in /shared/httpd/my-typo $ ln -s typo3/public htdocs
```

How does the directory structure look after symlinking:

```
devilbox@php-7.0.20 in /shared/httpd/my-typo $ tree -L 1
.
├─ typo3
└─ htdocs -> typo3/public

2 directories, 0 files
```

As you can see from the above directory structure, `htdocs` is available in its expected path and points to the frameworks entrypoint.

### 75.2.5 5. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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-typo.loc
```

See also:

- [Add project hosts entry on MacOS](#)
- [Add project hosts entry on Windows](#)

- *Setup Auto DNS*

### 75.2.6 6. Create `FIRST_INSTALL` file

To continue installing via the guided web install, you need to create a file called `FIRST_INSTALL` in the document root.

```
devilbox@php-7.0.20 in /shared/httpd/my-typo $ touch htdocs/FIRST_INSTALL
```

### 75.2.7 7. Open your browser

Open your browser at <http://my-typo.loc> or <https://my-typo.loc>.

**See also:**

*Setup valid HTTPS*

### 75.2.8 8. Step through guided web installation

1. Select database
  - Connection: Manually configured MySQL TCP/IP connection
  - Username: root
  - Password
  - Host: 127.0.0.1
  - Port: 3306
2. Select database
  - Create a new database: `typo3`
3. Create Administrative User / Specify Site Name
  - Username: admin
  - Password: choose a secure password
  - Site name: My Typo
4. Installation complete
  - Create empty starting page

---

## Setup Wordpress

---

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

After completing the below listed steps, you will have a working Laravel setup ready to be served via http and https.

**See also:**

**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. Add MySQL Database*
  - *6. DNS record*
  - *7. Open your browser*

### 76.1 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	<a href="http://my-wp.loc">http://my-wp.loc</a> <a href="https://my-wp.loc">https://my-wp.loc</a>

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via `HOST_PATH_HTTPD_DATADIR`.
- 

## 76.2 Walk through

It will be ready in seven simple steps:

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

### 76.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

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

### 76.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. ( `<vhost dir>.TLD_SUFFIX` will be the final URL ).

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

**See also:**

*[TLD\\_SUFFIX](#)*



### 76.2.3 3. Download Wordpress via git

Navigate into your newly created vhost directory and install Wordpress with 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
```

How does the directory structure look after installation:

```
devilbox@php-7.0.20 in /shared/httpd/my-wp $ tree -L 1
.
├── wordpress.git

1 directory, 0 files
```

### 76.2.4 4. Symlink webroot

Symlinking the actual webroot directory to `htdocs` is important. The web server expects every project's document root to be in `<vhost dir>/htdocs/`. This is the path where it will serve the files. This is also the path where your frameworks entrypoint (usually `index.php`) should be found.

Some frameworks however provide its actual content in nested directories of unknown levels. This would be impossible to figure out by the web server, so you manually have to symlink it back to its expected path.

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

How does the directory structure look after symlinking:

```
devilbox@php-7.0.20 in /shared/httpd/my-wp $ tree -L 1
.
├── wordpress.git
└── htdocs -> wordpress.git

2 directories, 0 files
```

As you can see from the above directory structure, `htdocs` is available in its expected path and points to the frameworks entrypoint.

### 76.2.5 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_wp;'
```

### 76.2.6 6. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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:**

- *Add project hosts entry on MacOS*
- *Add project hosts entry on Windows*
- *Setup Auto DNS*

## 76.2.7 7. Open your browser

Open your browser at <http://my-wp.loc> or <https://my-wp.loc> and follow the installation steps.

**See also:**

*Setup valid HTTPS*

**(1/7) Choose your desired Wordpress language**

**(2/7) Read pre-installation information**

**(3/7) Setup database connection**

---

**Important:** Choose `127.0.0.1` as the database host

---

**(4/7) Database setup post screen**

**(5/7) Start Wordpress installation**

**(6/7) Installation success view**

**(7/7) Login to Admin panel**

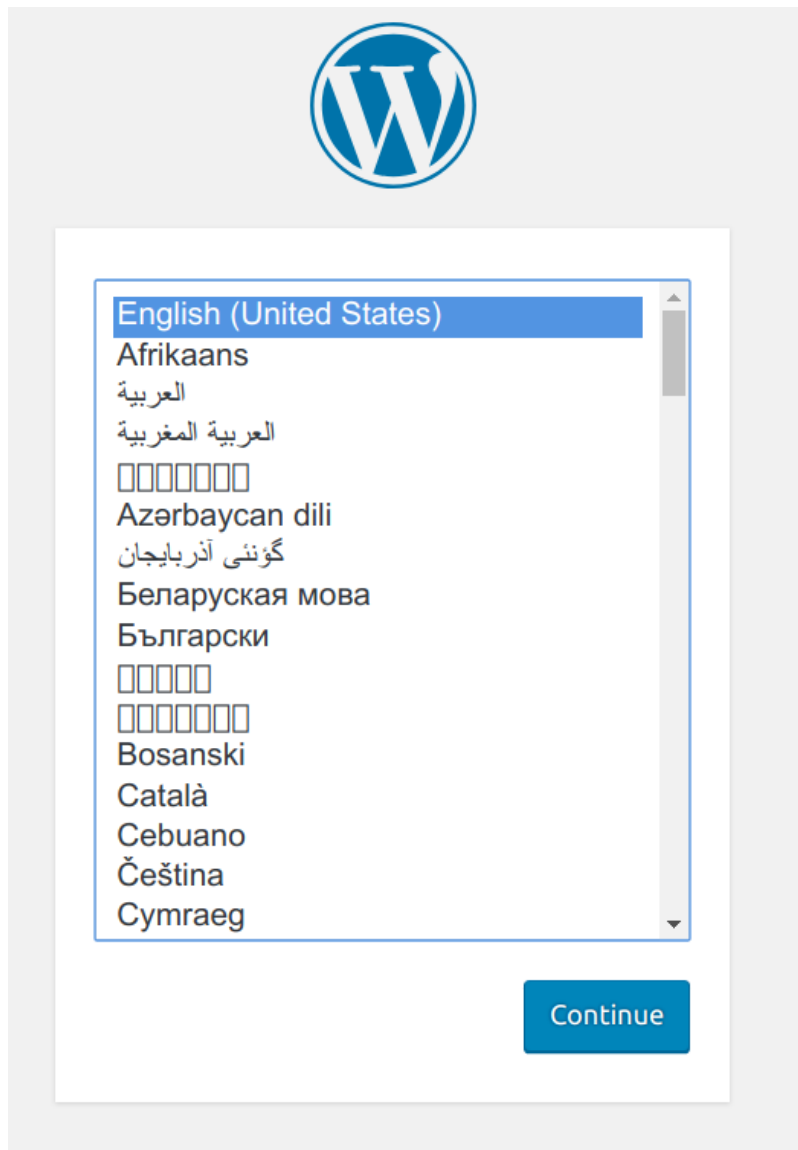


Fig. 1: Wordpress installation: Choose language

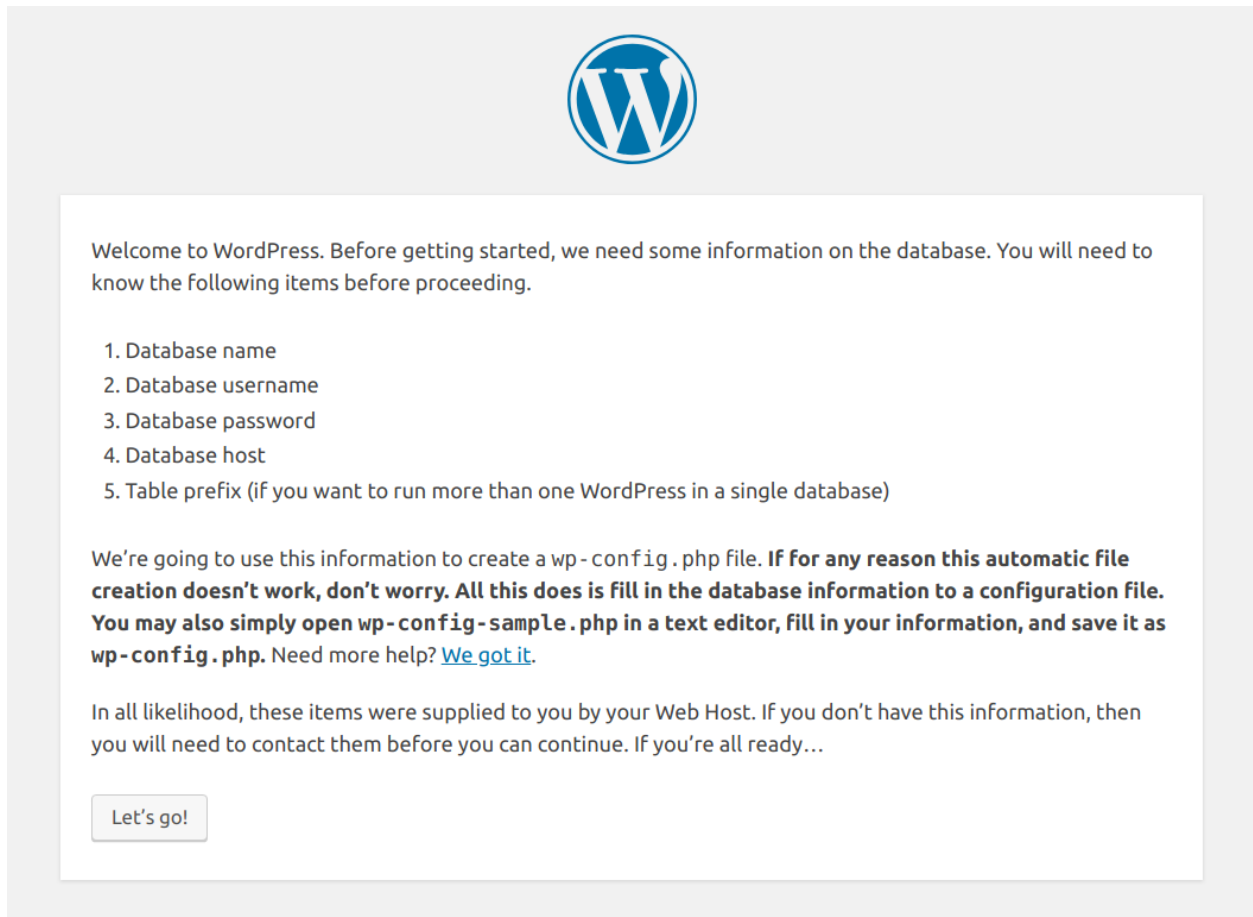




Fig. 2: Wordpress installation: Overview



Below you should enter your database connection details. If you're not sure about these, contact your host.


<b>Database Name</b>	<input type="text" value="my_wp"/>	The name of the database you want to use with WordPress.
<b>Username</b>	<input type="text" value="root"/>	Your database username.
<b>Password</b>	<input type="password"/>	Your database password.
<b>Database Host</b>	<input type="text" value="127.0.0.1"/>	You should be able to get this info from your web host, if localhost doesn't work.
<b>Table Prefix</b>	<input type="text" value="wp_"/>	If you want to run multiple WordPress installations in a single database, change this.

Fig. 3: Wordpress installation: Setup database



All right, sparky! You've made it through this part of the installation. WordPress can now communicate with your database. If you are ready, time now to...

Fig. 4: Wordpress installation: Database setup finished



## Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

## Information needed

Please provide the following information. Don't worry, you can always change these settings later.

Site Title	<input type="text" value="my wordpress"/>
Username	<input type="text" value="admin"/> <small>Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.</small>
Password	<div><input type="password" value="....."/> Strong</div> <div><input type="button" value="Show"/></div> <p><b>Important:</b> You will need this password to log in. Please store it in a secure location.</p>
Your Email	<input type="text" value="admin@my-wp.loc"/> <small>Double-check your email address before continuing.</small>
Search Engine Visibility	<input type="checkbox"/> Discourage search engines from indexing this site <small>It is up to search engines to honor this request.</small>

Fig. 5: Wordpress installation: Installation

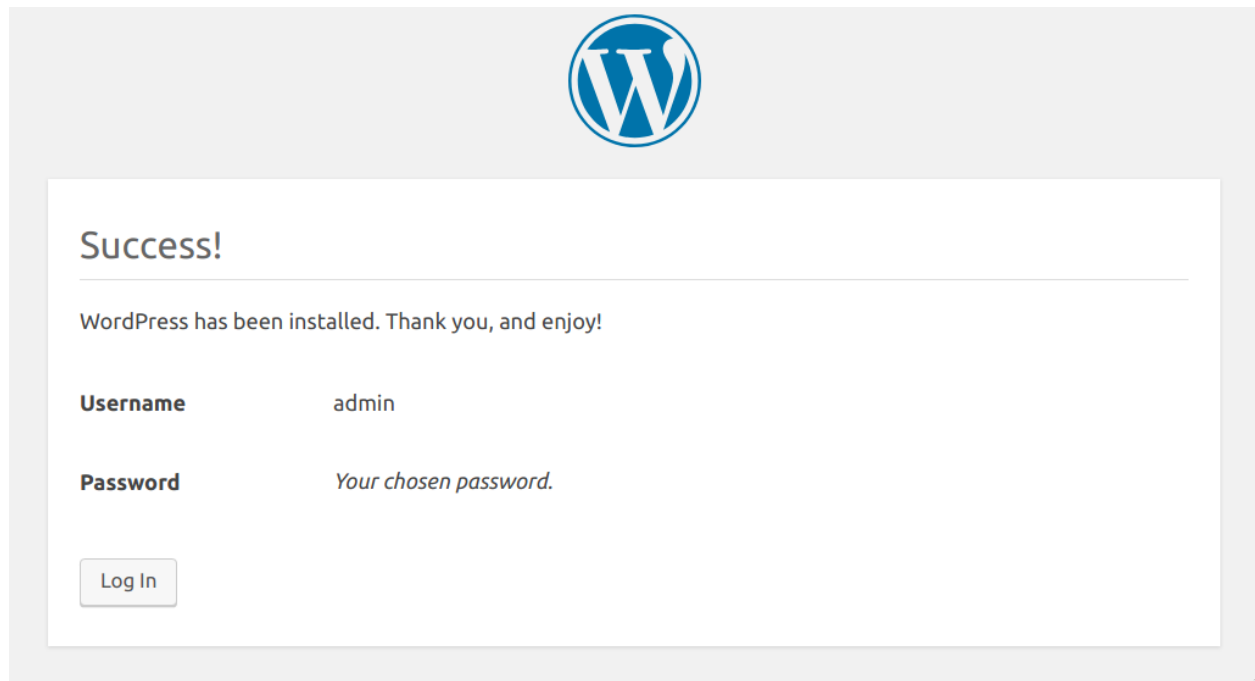


Fig. 6: Wordpress installation: Installation finished

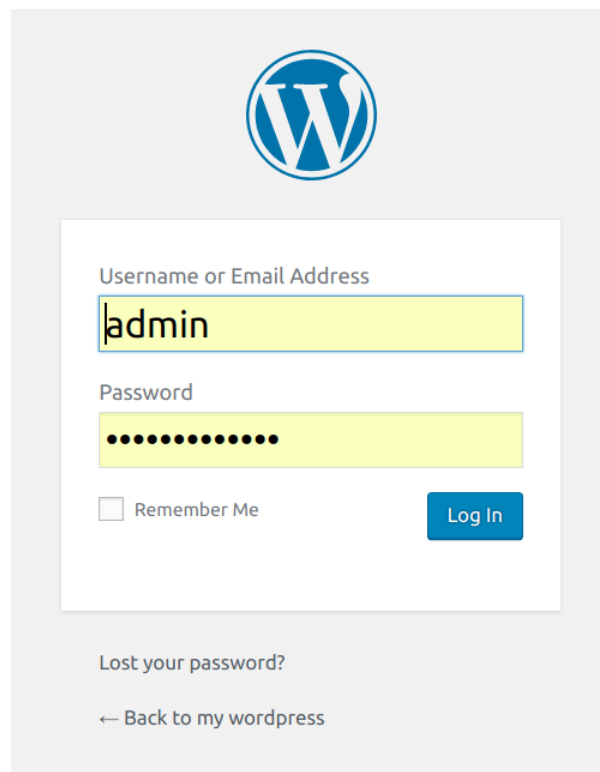


Fig. 7: Wordpress installation: Login





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

After completing the below listed steps, you will have a working Laravel setup ready to be served via http and https.

**See also:**

**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

## 77.1 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	<a href="http://my-yii.loc">http://my-yii.loc</a> <a href="https://my-yii.loc">https://my-yii.loc</a>

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via [\*HOST\\_PATH\\_HTTPD\\_DATADIR\*](#).
- 

## 77.2 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

### 77.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

- [\*Enter the PHP container\*](#)
- [\*Work inside the PHP container\*](#)
- [\*Available tools\*](#)

### 77.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. ( `<vhost dir>.TLD_SUFFIX` will be the final URL ).

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

**See also:**

[\*TLD\\_SUFFIX\*](#)

### 77.2.3 3. Install Yii2 via `composer`

Navigate into your newly created vhost directory and install Yii2 with `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
```

(continues on next page)

(continued from previous page)

How does the directory structure look after installation:

```
devilbox@php-7.0.20 in /shared/httpd/my-yii $ tree -L 1
.
└─ yii2-dev

1 directory, 0 files
```

#### 77.2.4 4. Symlink webroot

Symlinking the actual webroot directory to `htdocs` is important. The web server expects every project's document root to be in `<vhost dir>/htdocs/`. This is the path where it will serve the files. This is also the path where your frameworks entrypoint (usually `index.php`) should be found.

Some frameworks however provide its actual content in nested directories of unknown levels. This would be impossible to figure out by the web server, so you manually have to symlink it back to its expected path.

```
devilbox@php-7.0.20 in /shared/httpd/my-yii $ ln -s yii2-dev/web/ htdocs
```

How does the directory structure look after symlinking:

```
devilbox@php-7.0.20 in /shared/httpd/my-yii $ tree -L 1
.
├─ yii2-dev
└─ htdocs -> yii2-dev/web

2 directories, 0 files
```

As you can see from the above directory structure, `htdocs` is available in its expected path and points to the frameworks entrypoint.

#### 77.2.5 5. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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:

- [Add project hosts entry on MacOS](#)
- [Add project hosts entry on Windows](#)
- [Setup Auto DNS](#)

### 77.2.6 6. Open your browser

Open your browser at <http://my-yii.loc> or <https://my-yii.loc>

**See also:**

*Setup valid HTTPS*

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

**See also:**

**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*

## 78.1 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	<a href="http://my-zend.loc">http://my-zend.loc</a> <a href="https://my-zend.loc">https://my-zend.loc</a>

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.

- On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via `HOST_PATH_HTTPD_DATADIR`.
- 

## 78.2 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

### 78.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

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

### 78.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. ( `<vhost dir>.TLD_SUFFIX` will be the final URL ).

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

**See also:**

*[TLD\\_SUFFIX](#)*

### 78.2.3 3. Install Zend via `composer`

Navigate into your newly created vhost directory and install Zend with `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
```

How does the directory structure look after installation:

```
devilbox@php-7.0.20 in /shared/httpd/my-zend $ tree -L 1
.
└─ zend

1 directory, 0 files
```

## 78.2.4 4. Symlink webroot

Symlinking the actual webroot directory to `htdocs` is important. The web server expects every project's document root to be in `<vhost dir>/htdocs/`. This is the path where it will serve the files. This is also the path where your frameworks entypoint (usually `index.php`) should be found.

Some frameworks however provide its actual content in nested directories of unknown levels. This would be impossible to figure out by the web server, so you manually have to symlink it back to its expected path.

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

How does the directory structure look after symlinking:

```
devilbox@php-7.0.20 in /shared/httpd/my-zend $ tree -L 1
.
├─ zend
└─ htdocs -> zend/public

2 directories, 0 files
```

As you can see from the above directory structure, `htdocs` is available in its expected path and points to the frameworks entypoint.

## 78.2.5 5. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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:

- *Add project hosts entry on MacOS*
- *Add project hosts entry on Windows*
- *Setup Auto DNS*

## 78.2.6 6. Open your browser

Open your browser at <http://my-zend.loc> or <https://my-zend.loc>

See also:

*Setup valid HTTPS*



---

### Setup other Frameworks

---

The setup instructions and frameworks shown in this section are only meant as an example and the list is in no way complete.

The Devilbox itself is a normal development stack and is capable of running *any* framework, CMS or custom written PHP software.

If you wish to see more install guides, open up an issue at Github: <https://github.com/cytopia/devilbox/issues>



---

### Setup reverse proxy NodeJS

---

This example will walk you through creating a NodeJS hello world application, which is started automatically on `docker-compose up` via `up`, will be proxied to the web server and can be reached via valid HTTPS.

**Note:** It is also possible to attach a leight-weight NodeJS container to the Devilbox instead of running this in the PHP container. See here for details: [Reverse Proxy for custom Docker](#)

---

#### Table of Contents

- *Overview*
- *Walk through*
  - *1. Enter the PHP container*
  - *2. Create new vhost directory*
  - *3. Create NodeJS application*
  - *4. Create virtual docroot directory*
  - *5. Add reverse proxy vhost-gen config files*
    - \* *5.1 Create vhost-gen template directory*
    - \* *5.2 Copy vhost-gen templates*
    - \* *5.3 Adjust ports*
      - *5.3.1 Adjust Apache 2.2 template*
      - *5.3.2 Adjust Apache 2.4 template*
      - *5.3.3 Adjust Nginx template*
  - *6. Create autostart script*

- 7. *DNS record*
- 8. *Restart the Devilbox*
- 9. *Open your browser*
- *Managing NodeJS*

## 80.1 Overview

The following configuration will be used:

Project name	VirtualHost directory	Database	TLD_SUFFIX	Project URL
my-node	/shared/httpd/my-node	.	loc	<a href="http://my-node.loc">http://my-node.loc</a> <a href="https://my-node.loc">https://my-node.loc</a>

Additionally we will set the listening port of the NodeJS application to 4000 inside the PHP container.

We also want NodeJS running regardless of which PHP container will be started (global autostart).

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via `HOST_PATH_HTTPD_DATADIR`.
- 

## 80.2 Walk through

It will be ready in nine simple steps:

1. Enter the PHP container
2. Create a new VirtualHost directory
3. Create NodeJS hello world application
4. Create *virtual* docroot directory
5. Add reverse proxy vhost-gen config files
6. Create autostart script
7. Setup DNS record
8. Restart the Devilbox
9. Visit <http://my-node.loc> in your browser

### 80.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

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

### 80.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. ( `<vhost dir>.TLD_SUFFIX` will be the final URL ).

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

**See also:**

[\*TLD\\_SUFFIX\*](#)

### 80.2.3 3. Create NodeJS application

```
# Navigate to your project directory
devilbox@php-7.0.20 in /shared/httpd $ cd my-node

# Create a directory which will hold the source code
devilbox@php-7.0.20 in /shared/httpd/my-node $ mkdir src

# Create the index.js file with your favourite editor
devilbox@php-7.0.20 in /shared/httpd/my-node/src $ vi index.js
```

Listing 1: index.js

```
// Load the http module to create an http server.
var http = require('http');

// Configure our HTTP server to respond with Hello World to all requests.
var server = http.createServer(function (request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.end("Hello World\n");
});

// Listen on port 3000
server.listen(3000);
```

## 80.2.4 4. Create *virtual* docroot directory

Every project for the Devilbox requires a `htdocs` directory present inside the project dir. For a reverse proxy this is not of any use, but rather only for the Intranet vhost page to stop complaining about the missing `htdocs` directory. So that's why this is only a *virtual* directory which will not hold any data.

```
# Navigate to your project directory
devilbox@php-7.0.20 in /shared/httpd $ cd my-node

# Create the docroot directory
devilbox@php-7.0.20 in /shared/httpd/my-node $ mkdir htdocs
```

See also:

*HTTPD\_DOCROOT\_DIR*

## 80.2.5 5. Add reverse proxy vhost-gen config files

### 5.1 Create vhost-gen template directory

Before we can copy the vhost-gen templates, we must create the `.devilbox` template directory inside the project directory.

```
# Navigate to your project directory
devilbox@php-7.0.20 in /shared/httpd $ cd my-node

# Create the .devilbox template directory
devilbox@php-7.0.20 in /shared/httpd/my-node $ mkdir .devilbox
```

See also:

*HTTPD\_TEMPLATE\_DIR*

### 5.2 Copy vhost-gen templates

Now we can copy and adjust the vhost-gen reverse proxy files for Apache 2.2, Apache 2.4 and Nginx.

The reverse vhost-gen templates are available in `cfg/vhost-gen`:

```
host> tree -L 1 cfg/vhost-gen/

cfg/vhost-gen/
├── apache22.yml-example-rproxy
├── apache22.yml-example-vhost
├── apache24.yml-example-rproxy
├── apache24.yml-example-vhost
├── nginx.yml-example-rproxy
├── nginx.yml-example-vhost
└── README.md

0 directories, 7 files
```

For this example we will copy all `*-example-rproxy` files into `/shared/httpd/my-node/.devilbox` to ensure this will work with all web servers.

```
host> cd /path/to/devilbox
host> cp cfg/vhost-gen/apache22.yml-example-rproxy data/www/my-node/.devilbox/
↪ apache22.yml
host> cp cfg/vhost-gen/apache24.yml-example-rproxy data/www/my-node/.devilbox/
↪ apache24.yml
host> cp cfg/vhost-gen/nginx.yml-example-rproxy data/www/my-node/.devilbox/nginx.yml
```

## 5.3 Adjust ports

By default, all vhost-gen templates will forward requests to port 8000 into the PHP container. Our current example however uses port 4000, so we must change that accordingly for all three templates.

### 5.3.1 Adjust Apache 2.2 template

Open the `apache22.yml` vhost-gen template in your project:

```
host> cd /path/to/devilbox
host> vi data/www/my-node/.devilbox/apache22.yml
```

Find the two lines with `ProxyPass` and `ProxyPassReverse` and change the port from 8000 to 4000

Listing 2: `data/www/my-node/.devilbox/apache22.yml`

```
# ... more lines above ... #

###
### Basic vHost skeleton
###
vhost: |
  <VirtualHost __DEFAULT_VHOST__:__PORT__>
    ServerName __VHOST_NAME__

    CustomLog "__ACCESS_LOG__" combined
    ErrorLog "__ERROR_LOG__"

    # Reverse Proxy definition (Ensure to adjust the port, currently '8000')
    ProxyRequests On
    ProxyPreserveHost On
    ProxyPass / http://php:4000/
    ProxyPassReverse / http://php:4000/

# ... more lines below ... #
```

### 5.3.2 Adjust Apache 2.4 template

Open the `apache24.yml` vhost-gen template in your project:

```
host> cd /path/to/devilbox
host> vi data/www/my-node/.devilbox/apache24.yml
```

Find the two lines with `ProxyPass` and `ProxyPassReverse` and change the port from 8000 to 4000

Listing 3: data/www/my-node/.devilbox/apache24.yml

```
# ... more lines above ... #

###
### Basic vHost skeleton
###
vhost: |
  <VirtualHost __DEFAULT_VHOST__:__PORT__>
    ServerName    __VHOST_NAME__

    CustomLog      "__ACCESS_LOG__" combined
    ErrorLog       "__ERROR_LOG__"

    # Reverse Proxy definition (Ensure to adjust the port, currently '8000')
    ProxyRequests On
    ProxyPreserveHost On
    ProxyPass       / http://php:4000/
    ProxyPassReverse / http://php:4000/

# ... more lines below ... #
```

### 5.3.3 Adjust Nginx template

Open the `nginx.yml` vhost-gen template in your project:

```
host> cd /path/to/devilbox
host> vi data/www/my-node/.devilbox/nginx.yml
```

Find the lines with `proxy_pass` and change the port from 8000 to 4000

Listing 4: data/www/my-node/.devilbox/nginx.yml

```
# ... more lines above ... #

###
### Basic vHost skeleton
###
vhost: |
  server {
    listen        __PORT__ __DEFAULT_VHOST__;
    server_name    __VHOST_NAME__;

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

    # Reverse Proxy definition (Ensure to adjust the port, currently '8000')
    location / {
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_pass http://php:4000;
    }

# ... more lines below ... #
```



## 80.2.6 6. Create autostart script

For NodeJS applications, the Devilbox already bundles an autostart template which you can use and simply just add the path of your NodeJS application. This template does nothing by default as its file name does not end by .sh. So let's have a look at the template from autostart/run-node-js-projects.sh-example. The location where you will have to add your path is highlighted:

Listing 5: autostart/run-node-js-projects.sh-example

```
#!/usr/bin/env bash
#
# This is a generic example to startup your NodeJS projects with
# pm2 (https://github.com/Unitech/pm2)
#
# Important: As everything is run by the root user, you must explicitly direct the
#           commands to the devilbox user.
#

# Add the full paths of your Nodejs projects startup files into this array
# Each project separated by a newline and enclosed in double quotes. (No commas!)
# Paths are internal paths inside the PHP container.
NODE_PROJECTS=(
    #"/shared/httpd/my-rhost/js/index.js"
    #"/shared/httpd/my-node-hello-world/name/run.js"
    #"/shared/httpd/another-node-project/javascript/run.js"
)

# Check if any projects have been defined
if [ ${#NODE_PROJECTS[@]} -eq 0 ]; then
    echo "No projects defined. Exiting."
    exit 0
fi

# This loops over the paths, separates base directory and filename and will run it in_
↳ the background
# as the user devilbox. There shouldn't be any need to change anything here.
for item in ${NODE_PROJECTS[*]}; do
    NODE_PATH="$( dirname "${item}" )"
    NODE_FILE="$( basename "${item}" )"

    if [ ! -d "${NODE_PATH}" ]; then
        >&2 echo "[Warning], skipping startup, directory does not exist: $
↳ ${NODE_PATH}"
        continue;
    fi
    if [ ! -f "${NODE_PATH}/${NODE_FILE}" ]; then
        >&2 echo "[Warning], skipping startup, file does not exist: ${NODE_
↳ PATH}/${NODE_FILE}"
        continue;
    fi

    echo "su -c \"cd ${NODE_PATH}; pm2 start ${NODE_FILE}\" -l devilbox"
    su -c "cd ${NODE_PATH}; pm2 start ${NODE_FILE}" -l devilbox
done
```

So in order to proceed copy this file inside the `autostart/` directory of the Devilbox git directory to a new file ending by `.sh`

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

# Navigate to the autostart directory
host> cd autostart

# Copy the template
host> cp run-node-js-projects.sh-example run-node-js-projects.sh

# Adjust the template and add your path:
host> vi run-node-js-projects.sh
```

Listing 6: `autostart/run-node-js-projects.sh`

```
# ... more lines above ... #

# Add the full paths of your Nodejs projects startup files into this array
# Each project separated by a newline and enclosed in double quotes. (No commas!)
# Paths are internal paths inside the PHP container.
NODE_PROJECTS=(
    "/shared/httpd/my-node/js/index.js"
)

# ... more lines below ... #
```

**See also:**

- *Custom scripts per PHP version* (individually for different PHP versions)
- *Custom scripts globally* (equal for all PHP versions)
- *Autostarting NodeJS Apps*

### 80.2.7 7. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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 7: `/etc/hosts`

```
127.0.0.1 my-node.loc
```

**See also:**

- *Add project hosts entry on MacOS*
- *Add project hosts entry on Windows*
- *Setup Auto DNS*

### 80.2.8 8. Restart the Devilbox

Now for those changes to take affect, you will have to restart the Devilbox.

```

host> cd /path/to/devilbox

# Stop the Devilbox
host> docker-compose down
host> docker-compose rm -f

# Start the Devilbox
host> docker-compose up -d php httpd bind

```

### 80.2.9 9. Open your browser

All set now, you can visit <http://my-node.loc> or <https://my-node.loc> in your browser. The NodeJS application has been started up automatically and the reverse proxy will direct all requests to it.

**See also:**

*Setup valid HTTPS*

## 80.3 Managing NodeJS

If you have never worked with , I suggest to visit their website and get familiar with the available commands. A quick guide is below:

```

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

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

# List your running NodeJS apps
devilbox@php-7.0.20 in /shared/httpd $ pm2 list

```

App name	id	version	mode	pid	status	restart	uptime	cpu	mem
→ user	watching								
index	0	N/A	fork	1906	online	0	42h	0%	39.7 MB
→ devilbox	disabled								



---

## Setup reverse proxy Sphinx docs

---

This example will walk you through creating a Sphinx documentation, which is started automatically on `docker-compose up`, will be proxied to the web server and can be reached via valid HTTPS.

---

**Note:** It is also possible to attach a leight-weight Python container to the Devilbox instead of running this in the PHP container. See here for details: [\*Reverse Proxy for custom Docker\*](#)

---

### Table of Contents

- *Overview*
- *Walk through*
  - *1. Enter the PHP container*
  - *2. Create new vhost directory*
  - *3. Create basic Sphinx project*
  - *4. Create virtual docroot directory*
  - *5. Add reverse proxy vhost-gen config files*
    - \* *5.1 Create vhost-gen template directory*
    - \* *5.2 Copy vhost-gen templates*
    - \* *5.3 Adjust ports*
      - *5.3.1 Adjust Apache 2.2 template*
      - *5.3.2 Adjust Apache 2.4 template*
      - *5.3.3 Adjust Nginx template*
  - *6. Create autostart script*

- 7. *DNS record*
- 8. *Restart the Devilbox*
- 9. *Open your browser*

## 81.1 Overview

The following configuration will be used:

Project name	VirtualHost directory	Database	TLD_SUFFIX	Project URL
my-sphinx	/shared/httpd/my-sphinx	.	loc	<a href="http://my-sphinx.loc">http://my-sphinx.loc</a> <a href="https://my-sphinx.loc">https://my-sphinx.loc</a>

Additionally we will set the listening port of the Sphinx application to 4000 inside the PHP container.

We also want Sphinx running and autostarted only in the PHP 7.2 container (local autostart) and have all its required Python packages installed during `docker-compose up`.

---

**Note:**

- Inside the Devilbox PHP container, projects are always in `/shared/httpd/`.
  - On your host operating system, projects are by default in `./data/www/` inside the Devilbox git directory. This path can be changed via `HOST_PATH_HTTPD_DATADIR`.
- 

## 81.2 Walk through

It will be ready in nine simple steps:

1. Enter the PHP container
2. Create a new VirtualHost directory
3. Create basic Sphinx project
4. Create *virtual* docroot directory
5. Add reverse proxy vhost-gen config files
6. Create autostart script
7. Setup DNS record
8. Restart the Devilbox
9. Visit <http://my-sphinx.loc> in your browser

### 81.2.1 1. Enter the PHP container

All work will be done inside the PHP container as it provides you with all required command line tools.

Navigate to the Devilbox git directory and execute `shell.sh` (or `shell.bat` on Windows) to enter the running PHP container.

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

**See also:**

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

### 81.2.2 2. Create new vhost directory

The vhost directory defines the name under which your project will be available. ( `<vhost dir>.TLD_SUFFIX` will be the final URL ).

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

**See also:**

*TLD\_SUFFIX*

### 81.2.3 3. Create basic Sphinx project

```
# Navigate to your project directory
devilbox@php-7.0.20 in /shared/httpd $ cd my-sphinx

# Create a directory which will hold the documentation source code
devilbox@php-7.0.20 in /shared/httpd/my-sphinx $ mkdir doc
```

Create a basic Sphinx configuration file:

Listing 1: `/shared/httpd/my-sphinx/doc/conf.py`

```
source_suffix = '.rst'
master_doc = 'index'
html_theme = 'default'

exclude_patterns = [
    u'_build/*'
]
```

Create the table of contents file:

Listing 2: `/shared/httpd/my-sphinx/doc/index.rst`

```
.. :hidden:

*****
My Docs
```

(continues on next page)

(continued from previous page)

```
*****  
  
Description  
  
.. toctree::  
    :maxdepth: 2  
  
    page1
```

Create the first page `page1`:

Listing 3: `/shared/httpd/my-sphinx/doc/page1.rst`

```
*****  
Page 1  
*****  
  
Hello world
```

## 81.2.4 4. Create *virtual* docroot directory

Every project for the Devilbox requires a `htdocs` directory present inside the project dir. For a reverse proxy this is not of any use, but rather only for the Intranet vhost page to stop complaining about the missing `htdocs` directory. So that's why this is only a *virtual* directory which will not hold any data.

```
# Navigate to your project directory  
devilbox@php-7.0.20 in /shared/httpd $ cd my-sphinx  
  
# Create the docroot directory  
devilbox@php-7.0.20 in /shared/httpd/my-sphinx $ mkdir htdocs
```

See also:

[\*HTTPD\\_DOCROOT\\_DIR\*](#)

## 81.2.5 5. Add reverse proxy vhost-gen config files

### 5.1 Create vhost-gen template directory

Before we can copy the vhost-gen templates, we must create the `.devilbox` template directory inside the project directory.

```
# Navigate to your project directory  
devilbox@php-7.0.20 in /shared/httpd $ cd my-sphinx  
  
# Create the .devilbox template directory  
devilbox@php-7.0.20 in /shared/httpd/my-sphinx $ mkdir .devilbox
```

See also:

[\*HTTPD\\_TEMPLATE\\_DIR\*](#)



## 5.2 Copy vhost-gen templates

Now we can copy and adjust the vhost-gen reverse proxy files for Apache 2.2, Apache 2.4 and Nginx.

The reverse vhost-gen templates are available in `cfg/vhost-gen`:

```
host> tree -L 1 cfg/vhost-gen/

cfg/vhost-gen/
├── apache22.yml-example-rproxy
├── apache22.yml-example-vhost
├── apache24.yml-example-rproxy
├── apache24.yml-example-vhost
├── nginx.yml-example-rproxy
├── nginx.yml-example-vhost
└── README.md

0 directories, 7 files
```

For this example we will copy all `*-example-rproxy` files into `/shared/httpd/my-sphinx/.devilbox` to ensure this will work with all web servers.

```
host> cd /path/to/devilbox
host> cp cfg/vhost-gen/apache22.yml-example-rproxy data/www/my-sphinx/.devilbox/
↪ apache22.yml
host> cp cfg/vhost-gen/apache24.yml-example-rproxy data/www/my-sphinx/.devilbox/
↪ apache24.yml
host> cp cfg/vhost-gen/nginx.yml-example-rproxy data/www/my-sphinx/.devilbox/nginx.yml
```

## 5.3 Adjust ports

By default, all vhost-gen templates will forward requests to port 8000 into the PHP container. Our current example however uses port 4000, so we must change that accordingly for all three templates.

### 5.3.1 Adjust Apache 2.2 template

Open the `apache22.yml` vhost-gen template in your project:

```
host> cd /path/to/devilbox
host> vi data/www/my-sphinx/.devilbox/apache22.yml
```

Find the two lines with `ProxyPass` and `ProxyPassReverse` and change the port from 8000 to 4000

Listing 4: `data/www/my-sphinx/.devilbox/apache22.yml`

```
# ... more lines above ... #

###
### Basic vHost skeleton
###
vhost: |
  <VirtualHost __DEFAULT_VHOST__:__PORT__>
    ServerName __VHOST_NAME__

    CustomLog "__ACCESS_LOG__" combined
```

(continues on next page)

(continued from previous page)

```
ErrorLog    "__ERROR_LOG__"

# Reverse Proxy definition (Ensure to adjust the port, currently '8000')
ProxyRequests On
ProxyPreserveHost On
ProxyPass / http://php:4000/
ProxyPassReverse / http://php:4000/

# ... more lines below ... #
```

### 5.3.2 Adjust Apache 2.4 template

Open the `apache24.yml` vhost-gen template in your project:

```
host> cd /path/to/devilbox
host> vi data/www/my-sphinx/.devilbox/apache24.yml
```

Find the two lines with `ProxyPass` and `ProxyPassReverse` and change the port from 8000 to 4000

Listing 5: `data/www/my-sphinx/.devilbox/apache24.yml`

```
# ... more lines above ... #

###
### Basic vHost skeleton
###
vhost: |
  <VirtualHost __DEFAULT_VHOST__:__PORT__>
    ServerName __VHOST_NAME__

    CustomLog "__ACCESS_LOG__" combined
    ErrorLog  "__ERROR_LOG__"

    # Reverse Proxy definition (Ensure to adjust the port, currently '8000')
    ProxyRequests On
    ProxyPreserveHost On
    ProxyPass / http://php:4000/
    ProxyPassReverse / http://php:4000/

# ... more lines below ... #
```

### 5.3.3 Adjust Nginx template

Open the `nginx.yml` vhost-gen template in your project:

```
host> cd /path/to/devilbox
host> vi data/www/my-sphinx/.devilbox/nginx.yml
```

Find the lines with `proxy_pass` and change the port from 8000 to 4000

Listing 6: data/www/my-sphinx/.devilbox/nginx.yml

```
# ... more lines above ... #

###
### Basic vHost skeleton
###
vhost: |
  server {
    listen      __PORT__DEFAULT_VHOST__;
    server_name __VHOST_NAME__;

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

    # Reverse Proxy definition (Ensure to adjust the port, currently '8000')
    location / {
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_pass http://php:4000;
    }

# ... more lines below ... #
```

### 81.2.6 6. Create autostart script

Remember, we only wanted to have our Sphinx application run on the PHP 7.2 container, so we will create a autostart script only for that container.

Navigate to `cfg/php-startup-7.2/` in the Devilbox git directory and create a new shell script ending by `.sh`

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

# Nagivate to startup directory for PHP 7.2 and create the script
host> cd cfg/php-startup-7.2/
host> vi my-sphinx.sh
```

Listing 7: cfg/php-startup-7.2/my-sphinx.sh

```
#!/usr/bin/env bash

# Install required Python modules as root user
pip install sphinx sphinx-autobuild

# Autostart Sphinx by devilbox user on Port 4000 and sent it to backgroun with '&'
sh -c "cd /shared/httpd/my-sphinx; sphinx-autobuild . _build/html -p 4000 -H 0.0.0.0" &
↪-l devilbox &
```

#### See also:

- *Custom scripts per PHP version* (individually for different PHP versions)
- *Custom scripts globally* (equal for all PHP versions)
- *Autostarting NodeJS Apps*

### 81.2.7 7. DNS record

If you **have** Auto DNS configured already, you can skip this section, because DNS entries will be available automatically by the bundled DNS server.

If you **don't 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 8: `/etc/hosts`

```
127.0.0.1 my-node.loc
```

**See also:**

- [Add project hosts entry on MacOS](#)
- [Add project hosts entry on Windows](#)
- [Setup Auto DNS](#)

### 81.2.8 8. Restart the Devilbox

Now for those changes to take affect, you will have to restart the Devilbox.

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

# Stop the Devilbox
host> docker-compose down
host> docker-compose rm -f

# Start the Devilbox
host> docker-compose up -d php httpd bind
```

### 81.2.9 9. Open your browser

All set now, you can visit <http://my-sphinx.loc> or <https://my-sphinx.loc> in your browser. The Sphinx application has been started up automatically and the reverse proxy will direct all requests to it.

**See also:**

[Setup valid HTTPS](#)

---

## 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.

### 82.1 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]
-----		
↪-----		
\$ ls -l		\$ ls -l
-rw-r--r-- user group index.php		-rw-r--r-- 1000 1000 index.php
drwxrwxrwx user group tmp/		drwxrwxrwx 1000 1000 tmp/

Your web application might now have created some temporary files (via the PHP-FPM process) inside the `tmp/` directory:

[Host]		[Container]
-----		
↪-----		
\$ ls -l tmp/		\$ ls -l tmp/
-rw-r--r-- 96 96 _tmp_cache01.php		-rw-r--r-- www www _tmp_cache01.php
-rw-r--r-- 96 96 _tmp_cache02.php		-rw-r--r-- www www _tmp_cache01.php

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.

## 82.2 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.

## 82.3 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.

---

## Available container

---

---

**Note:**

*Start the Devilbox* Find out how to start some or all container.

---

The following tables give 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.

### 83.1 Core container

These container are well integrated into the Devilbox intranet and are considered core container:

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

### 83.2 Additional container

Additional container that are not yet integrated into the Devilbox intranet are also available. Those container come via `docker-compose.override.yml` and must explicitly be enabled. They are disabled by default to prevent accidentally starting too many container and making your computer unresponsive.

Container	Name	Hostname	IP Address
Blackfire	blackfire	blackfire	172.16.238.200
MailHog	mailhog	mailhog	172.16.238.201
RabbitMQ	rabbit	rabbit	172.16.238.210
Solr	solr	solr	172.16.238.220

**See also:**

- *Enable all additional container*
- *Enable and configure Blackfire*
- *Enable and configure MailHog*
- *Enable and configure RabbitMQ*
- *Enable and configure Solr*



---

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 PHP container*

The PHP container is your workhorse and these are your tools:

Binary	Tool
ansible	
various binaries	
brew	
codecept	
composer	
dep	
drush	
drupal	
eslint	
git	
git-flow	
gulp	
grunt	
jsonlint	
laravel	
linkcheck	
mdl	
mdlint	
mongo*	Various MongoDB client tools
mysql*	Various MySQL client tools
mysqldump-secure	
node	

Continued on next page

Table 1 – continued from previous page

Binary	Tool
npm	
pg*	Various PostgreSQL client tools
phalcon	
php-cs-fixer	
phpcs	
phpcbf	
phpunit	
photon	
pm2	
redis*	Various Redis client tools
sass	
scss-lint	
ssh	
symfony	
tig	
webpack	
wp	
yamllint	
yarn	

---

**Note:** If you are in need of other tools, open up an issue at [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](#)

This section will contain common problems and how to resolve them. It will grow over time once there are more issues reported.

**See also:**

- *How To*
- *FAQ*

**Important:**

*Update the Devilbox* Issues are constantly being fixed. Before attempting to spend too much time digging into your issue, make sure you are running the latest git changes and have pulled the latest Docker images.

Also keep in mind that configuration files might change, so ensure to diff the default ones against your currently active ones for added, removed or changed values.

**Table of Contents**

- *General*
  - *Sudden unexplained problems on Windows*
  - *Address already in use*
  - *Unable to finish Pulling as unauthorized: incorrect username or password*
  - *localhost or 127.0.0.1 not found*
  - *ERROR: Version in “./docker-compose.yml” is unsupported*
- *Performance*
  - *Performance issues on Docker for Mac*
- *DNS issues*

- *zone ‘localhost’: already exists previous definition*
- *SSL issues*
  - *unable to get local issuer certificate*
- *Web server issues*
  - *Warning: DocumentRoot [/var/www/default/htdocs/] does not exist*
  - *403 forbidden*
    - \* *File and directory permissions*
    - \* *Shared volumes*
  - *504 Gateway timeout*
- *PHP issues*
  - *Fatal error: Cannot redeclare go()*
- *Database issues*
  - *Invalid bind mount spec*
  - *[Warning] World-writable config file ‘etc/mysql/docker-default.d/my.cnf’ is ignored*

## 85.1 General

### 85.1.1 Sudden unexplained problems on Windows

In case something stopped working for no reason, check out other Docker container. If you experience similar issues as well, check for any unattended Windows updates or updates to Docker itself. If those exist, try to revert them and see if that was the cause.

I heard many bug stories from fellow Windows users so far. A good contact point for that is the Docker forum itself: <https://forums.docker.com/c/docker-for-windows>

A few general things you should always do before attempting to open up issues are:

#### 1. Used default settings from env-example

Try using the exact settings from `env-example` as variables might have been updated in git.

```
# Ensure everything is stopped
host> cp env-example .env
```

#### 2. Clean, updated and minimal start

```
# Ensure everything is stopped
host> docker-compose stop
host> docker-compose kill
host> docker-compose rm -f

# Ensure everything is updated
host> docker-compose pull

# Start again
host> docker-compose up php httpd bind
```

### 3. Reset Docker credentials:

As it might sound strange, this fix might indeed solve a lot of problems on Windows. Go to your Docker settings and reset your credentials.

### 4. Shared volumes:

Ensure all your Devilbox data (Devilbox directory and project directory) are within the volumes that are shared by Docker. If not add those in the Docker settings.

## 85.1.2 Address already in use

One of the Docker container wants to bind to a port on the host system which is already taken. Figure out what service is listening on your host system and shut it down or change the port of the affected service in the Devilbox `.env` file.

Some examples of common error messages:

```
Error starting userland proxy: Bind for 0.0.0.0:80: unexpected error (Failure_
↳EADDRINUSE)
```

## 85.1.3 Unable to finish Pulling as unauthorized: incorrect username or password

This error might occur if you are already logged into a different Docker repository. To fix this error, sign out of your currently logged in repository and try again.

**See also:**

<https://github.com/cytopia/devilbox/issues/223>

## 85.1.4 localhost or 127.0.0.1 not found

If you are using Docker Toolbox, the Devilbox intranet is not available on localhost or 127.0.0.1, but rather on the IP address of the Docker Toolbox machine.

**See also:**

*Find Docker Toolbox IP address*

## 85.1.5 ERROR: Version in “./docker-compose.yml” is unsupported

This simply means your Docker and/or Docker Compose versions are outdated.

**See also:**

*Prerequisites*

## 85.2 Performance

### 85.2.1 Performance issues on Docker for Mac

By default Docker for Mac has performance issues on mounted directories with a lot of files inside. To overcome this issue you can apply different kinds of caching options to the mount points.

**See also:**

- *OSX: Performance*
- *MOUNT\_OPTIONS*

## 85.3 DNS issues

### 85.3.1 zone 'localhost': already exists previous definition

```
bind_1 | /etc/bind/devilbox-wildcard_dns.localhost.conf:1:
zone 'localhost': already exists previous definition:
/etc/bind/named.conf.default-zones:10
```

This error occurs when using `localhost` as the *TLD\_SUFFIX*.

See also:

- *TLD\_SUFFIX*
- <https://github.com/cytopia/devilbox/issues/291>

## 85.4 SSL issues

### 85.4.1 unable to get local issuer certificate

```
Errors occurred when trying to connect to www.example.com:
cURL error 77: error setting certificate verify locations: CAfile: certificate ./ca/
↪cacert.pem CApath: /etc/ssl/certs
```

This issue might arise if you set *TLD\_SUFFIX* to an official top level domain such as `.com`. What happens is that the bundled DNS server does a catch-all on the TLD and redirects all name resolution to the Devilbox's PHP container IP address.

If you want to access `https://www.example.com` in that case, the request goes to the PHP container which does not have a valid SSL certificate for that domain.

**Do not use official TLD's** for *TLD\_SUFFIX*.

See also:

- *TLD\_SUFFIX*
- <https://github.com/cytopia/devilbox/issues/275>

## 85.5 Web server issues

### 85.5.1 Warning: DocumentRoot [/var/www/default/htdocs/] does not exist

This error is most likely to only occur on Docker for Windows and is just a result of not working volumes mounts.

See also:

<https://forums.docker.com/t/volume-mounts-in-windows-does-not-work/10693>

## 85.5.2 403 forbidden

This error might occur for the Devilbox intranet or custom created projects.

### File and directory permissions

One of the causes could be wrongly set file and directory permissions.

First ensure the cloned git directory is readable for users, groups and others.

For the Devilbox intranet, ensure the `.devilbox/` directory is readable for users, groups and others. Also check files and directories within.

For projects, ensure an `index.php` or `index.html` exists and that all files and directories are readable for users, groups and others.

### Shared volumes

This might additionally occur on MacOS or Windows due to the Devilbox and/or its projects not being in the standard location of Docker Shared volumes.

Check your Docker settings to allow shared volumes for the path of the Devilbox and its projects.

## 85.5.3 504 Gateway timeout

This error occurs when the upstream PHP-FPM server takes longer to execute a script, than the timeout value set in the web server for PHP-FPM to answer.

For that to fix one must increase the PHP-FPM/Proxy timeout settings in the `.env` file.  
*HTTPD\_TIMEOUT\_TO\_PHP\_FPM*

See also:

- *HTTPD\_TIMEOUT\_TO\_PHP\_FPM*
- <https://github.com/cytopia/devilbox/issues/280>
- <https://github.com/cytopia/devilbox/issues/234>

## 85.6 PHP issues

### 85.6.1 Fatal error: Cannot redeclare go()

If you encounter this error, it is most likely that your current project declares the PHP function `go()` and that you have enabled the `swoole` module which also provides an implementation of that function.

To mitigate that issue, make sure that the `swoole` module is disabled in `.env`.

See also:

- *PHP\_MODULES\_DISABLE*
- <https://github.com/getkirby/kirby/issues/643>

## 85.7 Database issues

### 85.7.1 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*

### 85.7.2 [Warning] World-writable config file `/etc/mysql/docker-default.d/my.cnf` is ignored

This warning might occur when using *Docker Toolbox and the Devilbox* 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.

```
# Navigate 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>



Find common questions and answers here.

**See also:**

- *How To*
- *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 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 Codeigniter?*
  - *Does it work with CraftCMS?*
  - *Does it work with Drupal?*
  - *Does it work with Joomla?*
  - *Does it work with Laravel?*
  - *Does it work with Magento?*
  - *Does it work with Phalcon?*
  - *Does it work with Photon CMS?*
  - *Does it work with PrestaShop?*
  - *Does it work with Shopware?*
  - *Does it work with Symfony?*
  - *Does it work with Typo3?*
  - *Does it work with Wordpress?*
  - *Does it work with Yii?*
  - *Does it work with Zend?*
  - *Does it work with other Frameworks?*

## 86.1 General

### 86.1.1 Are there any differences between native Docker and Docker Toolbox?

Yes, read *Docker Toolbox and the Devilbox* to find out more.

### 86.1.2 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.

### 86.1.3 Why are mounted PostgreSQL data directories separated by version?

See: *Why are mounted MySQL data directories separated by version?*

### 86.1.4 Why are mounted MongoDB data directories separated by version?

See: *Why are mounted MySQL data directories separated by version?*

### 86.1.5 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`

### 86.1.6 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.

### 86.1.7 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.

### 86.1.8 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>

## 86.2 Configuration

### 86.2.1 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.

### 86.2.2 Can I change php.ini?

Yes, `php.ini` directives can be changed for each PHP version separately. See `php.ini`

### 86.2.3 Can I change my.cnf?

Yes, `my.cnf` directives can be changed for each MySQL version separately. See `my.cnf`

## 86.2.4 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. Also do not use any official domain TLDs such as `com`, `net`, `org`, etc.

## 86.2.5 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](#).

## 86.2.6 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 [Setup Auto DNS](#) for detailed instructions.

# 86.3 Compatibility

## 86.3.1 Does it work with CakePHP?

Yes, see [Setup CakePHP](#)

## 86.3.2 Does it work with Codeigniter?

Yes, see [Setup CodeIgniter](#)

## 86.3.3 Does it work with CraftCMS?

Yes, see [Setup CraftCMS](#)

## 86.3.4 Does it work with Drupal?

Yes, see [Setup Drupal](#)

## 86.3.5 Does it work with Joomla?

Yes, see [Setup Joomla](#)

## 86.3.6 Does it work with Laravel?

Yes, see [Setup Laravel](#)

### **86.3.7 Does it work with Magento?**

Yes, see *Setup Magento 2*

### **86.3.8 Does it work with Phalcon?**

Yes, see *Setup Phalcon*

### **86.3.9 Does it work with Photon CMS?**

Yes, see *Setup Photon CMS*

### **86.3.10 Does it work with PrestaShop?**

Yes, see *Setup PrestaShop*

### **86.3.11 Does it work with Shopware?**

Yes, see *Setup Shopware*

### **86.3.12 Does it work with Symfony?**

Yes, see *Setup Symfony*

### **86.3.13 Does it work with Typo3?**

Yes, see *Setup Typo3*

### **86.3.14 Does it work with Wordpress?**

Yes, see *Setup Wordpress*

### **86.3.15 Does it work with Yii?**

Yes, see *Setup Yii*

### **86.3.16 Does it work with Zend?**

Yes, see *Setup Zend*

### **86.3.17 Does it work with other Frameworks?**

Yes, see *Setup other Frameworks*



The How to section gathers information about various topics, that might need to be done at some point throughout the installation and configuration.

**See also:**

- [FAQ](#)
- [Troubleshooting](#)

## 87.1 Add custom DNS server on Android

Adding custom DNS server on Android works out of the box for each connected Wi-Fi network separately. There is no need to install external Apps.

**Table of Contents**

- [Change DNS server in Android directly](#)
- [Change DNS server with Third-Party App](#)

### 87.1.1 Change DNS server in Android directly

1. Navigate to **Settings -> Wi-Fi**
2. **Press and hold** on the Wi-Fi network you want to change
3. Choose **Modify network**
4. Scroll down and click on **Advanced options**
5. Scroll down and click on **DHCP**
6. Click on **Static**

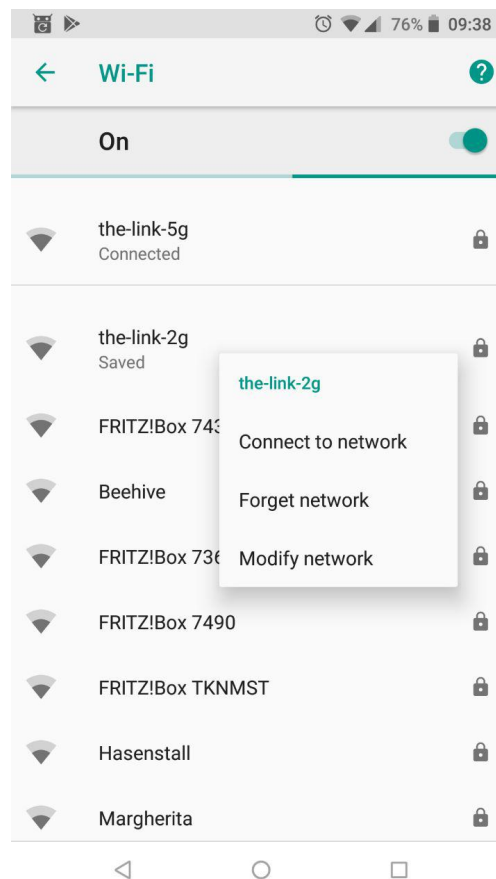


Fig. 1: Android: Wi-Fi list



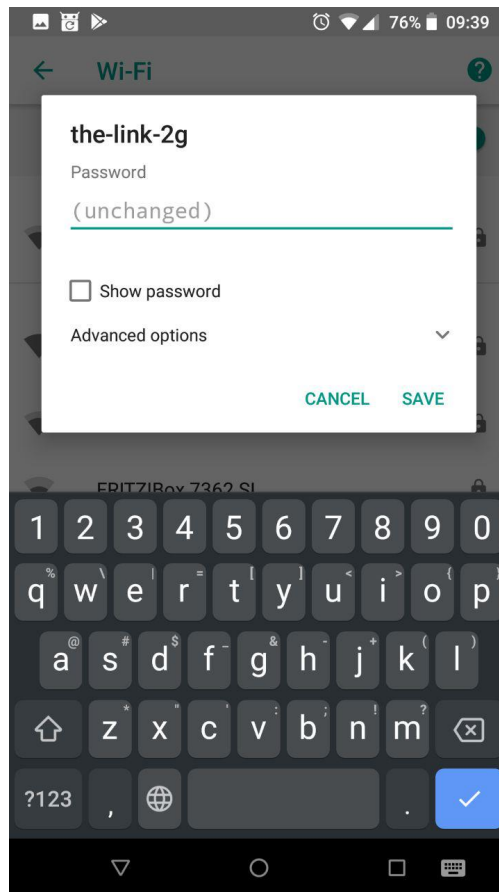


Fig. 2: Android: Advanced options

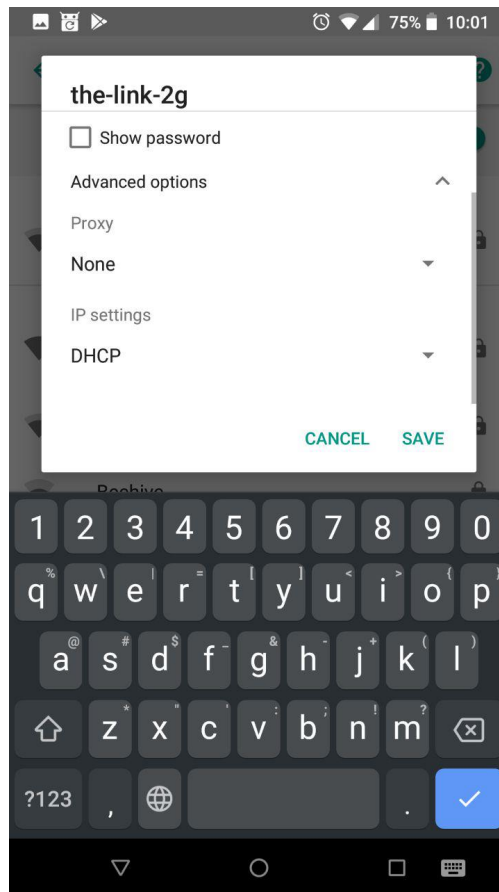


Fig. 3: Android: Select DHCP options

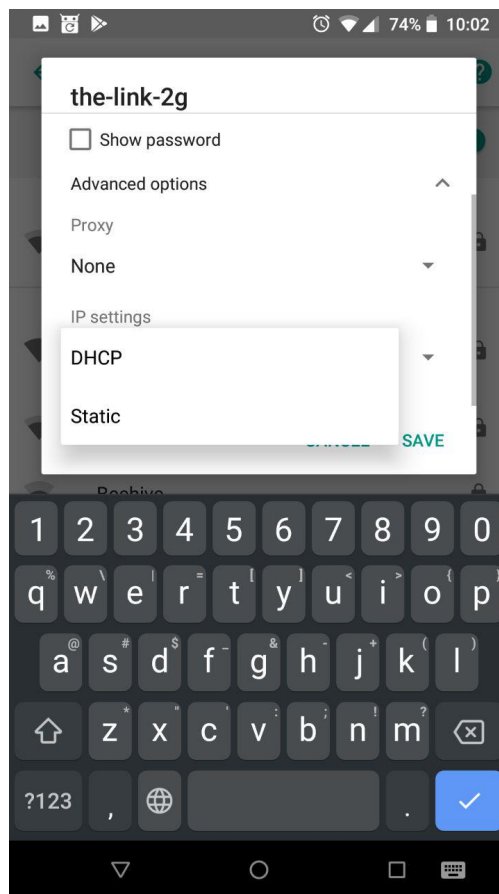


Fig. 4: Android: Select static DHCP options

7. Scroll down and change the DNS server IP for **DNS 1** (the first DNS server in the list)

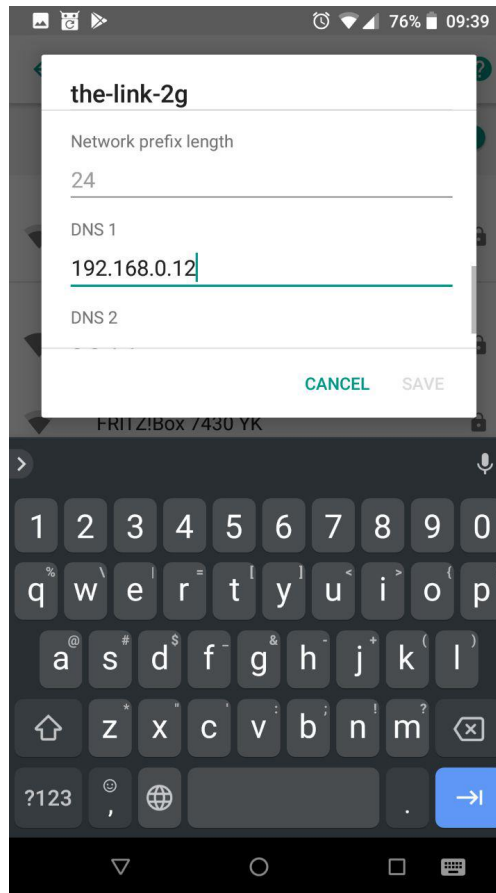


Fig. 5: Android: Add custom DNS server

### 87.1.2 Change DNS server with Third-Party App

If the above does not work for you or you just want another App that makes it even easier to change DNS settings, you can search the Playstore for many available DNS changer Apps. They also work on non-rooted Androids.

**See also:**

## 87.2 Add custom DNS server on iPhone

Adding custom DNS server on iPhone works out of the box for each connected Wi-Fi network separately. There is no need to install external Apps.

### Table of Contents

- [Change DNS server in iPhone directly](#)
- [Change DNS server with Third-Party App](#)

### 87.2.1 Change DNS server in iPhone directly

1. Navigate to **Settings** -> **Wi-Fi**
2. Tap on your active Wi-Fi connection

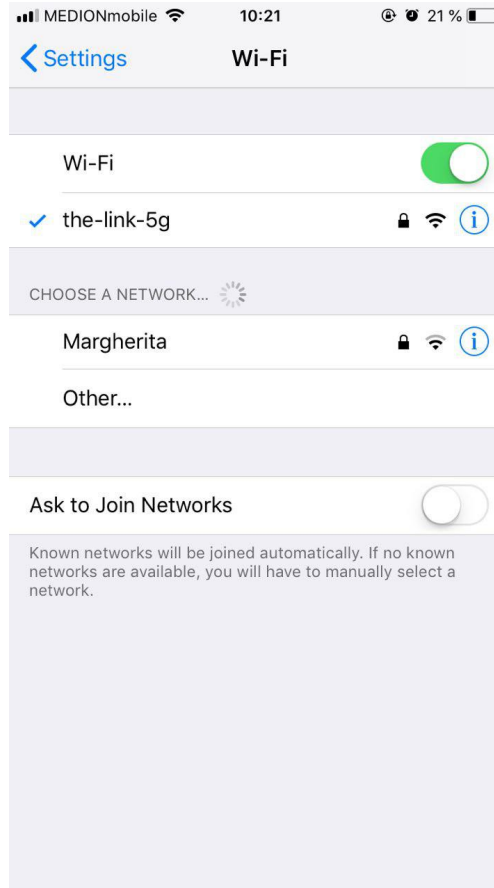


Fig. 6: iPhone: Wi-Fi list

3. Scroll down and tap on **Configure DNS**
4. Select **Manual**
5. Add your DNS server IP (ensure it is the first in the list)

### 87.2.2 Change DNS server with Third-Party App

If the above does not work for you or you just want another App that makes it even easier to change DNS settings, you can search the AppStore for many available DNS changer Apps. They also work on non-rooted iPhones.

**See also:**

## 87.3 Add custom DNS server on Linux

On Linux the DNS settings can be controlled by various different methods. Two of them are via Network Manager and systemd-resolved. Choose one of the methods depending on your local setup.

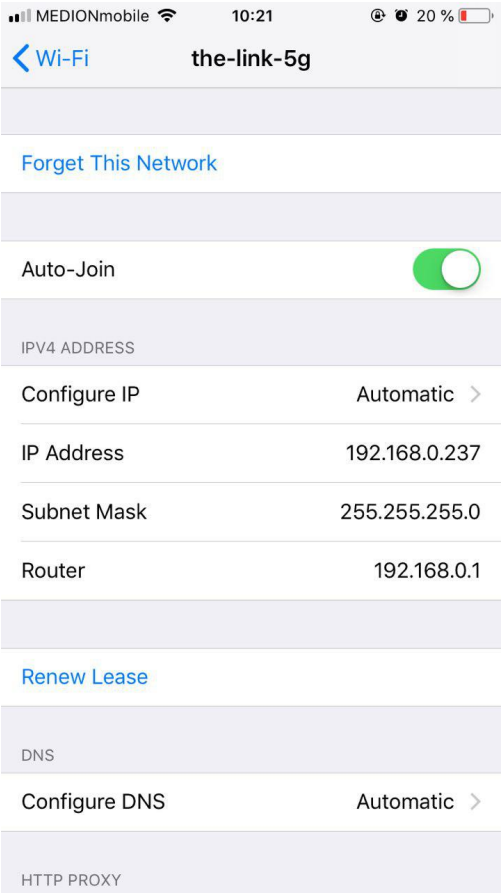


Fig. 7: iPhone: Wi-Fi list

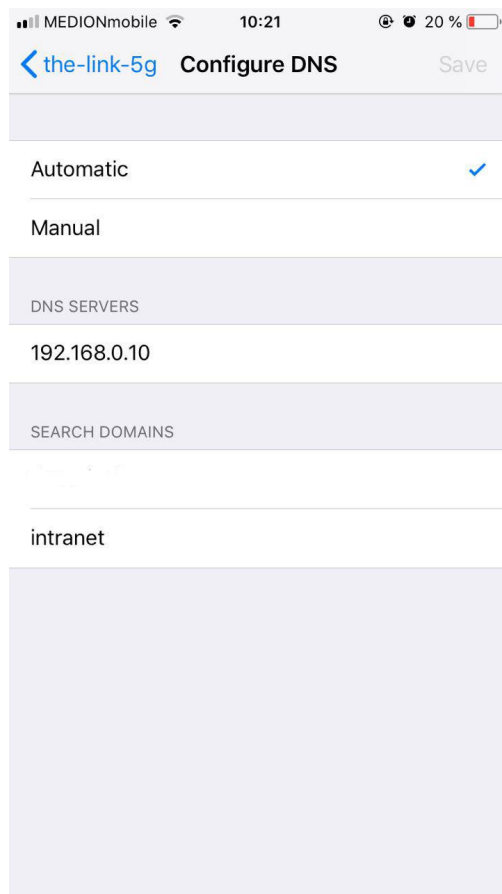


Fig. 8: iPhone: Wi-Fi list

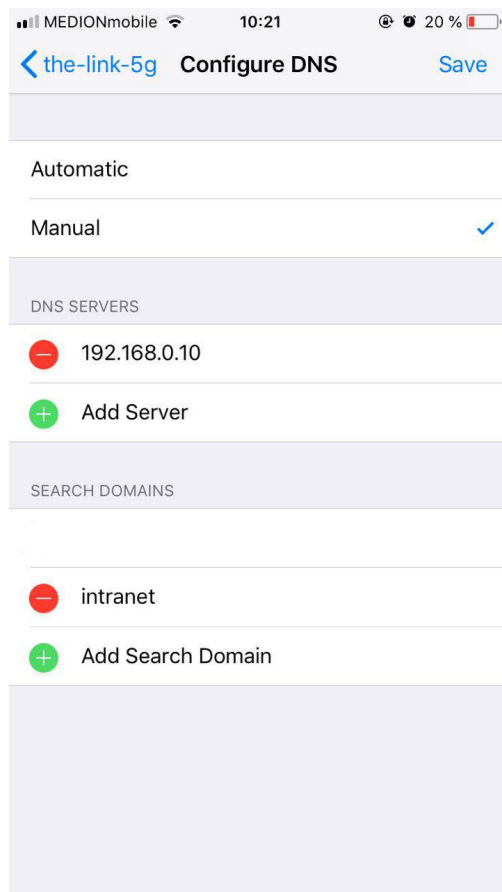


Fig. 9: iPhone: Wi-Fi list



## Table of Contents

- *Assumption*
- *Non permanent solution*
- *Network Manager*
- *systemd-resolved*

### 87.3.1 Assumption

This tutorial is using `127.0.0.1` as the DNS server IP address, as it is the method to setup Auto DNS for your local Devilbox.

### 87.3.2 Non permanent solution

When you just want to try out to add a new DNS server without permanent settings, you should use this option.

**Note:** Non permanent means, the settings will be gone when your DHCP release will be renewed, reconnecting to the network, restarting the network service, logging out or rebooting your machine.

1. Open `/etc/resolv.conf` with root or sudo privileges with your favourite editor on your host operating system:

```
host> sudo vi /etc/resolv.conf
```

2. Add your new `nameserver` directive **above** all existing `nameserver` directives:

Listing 1: `/etc/resolv.conf`

```
# Generated by NetworkManager
search intranet
nameserver 127.0.0.1
nameserver 192.168.0.10
```

3. It will work instantly after saving the file

### 87.3.3 Network Manager

*(This is a permanent solution and needs to be reverted when you don't need it anymore)*

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 (`127.0.0.1`).

Add the following line to to the very beginning of `/etc/dhcp/dhclient.conf`:

Listing 2: /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
host> sudo service network-manager restart

# Or the systemd way
host> 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`.

### 87.3.4 systemd-resolved

*(This is a permanent solution and needs to be reverted when you don't need it anymore)*

In case you are using `systemd-resolved` instead of `NetworkManager`, add the following line to the very beginning to `/etc/resolv.conf.head`:

Listing 3: /etc/resolv.conf.head

```
nameserver 127.0.0.1
```

Prevent `NetworkManager` from modifying `/etc/resolv.conf` and leave everything to `systemd-resolved` by adding the following line under the `[main]` section of `/etc/NetworkManager/NetworkManager.conf`

Listing 4: /etc/NetworkManager/NetworkManager.conf

```
dns=none
```

As a last step you will have to restart `systemd-resolved`.

```
host> sudo systemctl stop systemd-resolved
host> sudo systemctl start systemd-resolved
```

Once done, you can verify if the new DNS settings are effective:

```
host> systemd-resolve --status
```

See also:

## 87.4 Add custom DNS server on MacOS

### Table of Contents

- *Assumption*

- *Network preferences*

### 87.4.1 Assumption

This tutorial is using `127.0.0.1` as the DNS server IP address, as it is the method to setup Auto DNS for your local Devilbox.

### 87.4.2 Network preferences

1. Open System Preferences
2. Go to **Network**
3. Select your **active** connected interface
4. Click on **DNS** tab
5. Add new DNS server by clicking the **+** sign
6. Add `127.0.0.1` as the first entry

## 87.5 Add custom DNS server on Windows

### Table of Contents

- *Assumption*
- *Network preferences*

### 87.5.1 Assumption

This tutorial is using `127.0.0.1` as the DNS server IP address, as it is the method to setup Auto DNS for your local Devilbox.

### 87.5.2 Network preferences

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.

## 87.6 Add project hosts entry on Linux

On Linux, custom DNS entries can be added to the `/etc/hosts` and will take precedence over the same entries provided by any DNS server.

### Table of Contents

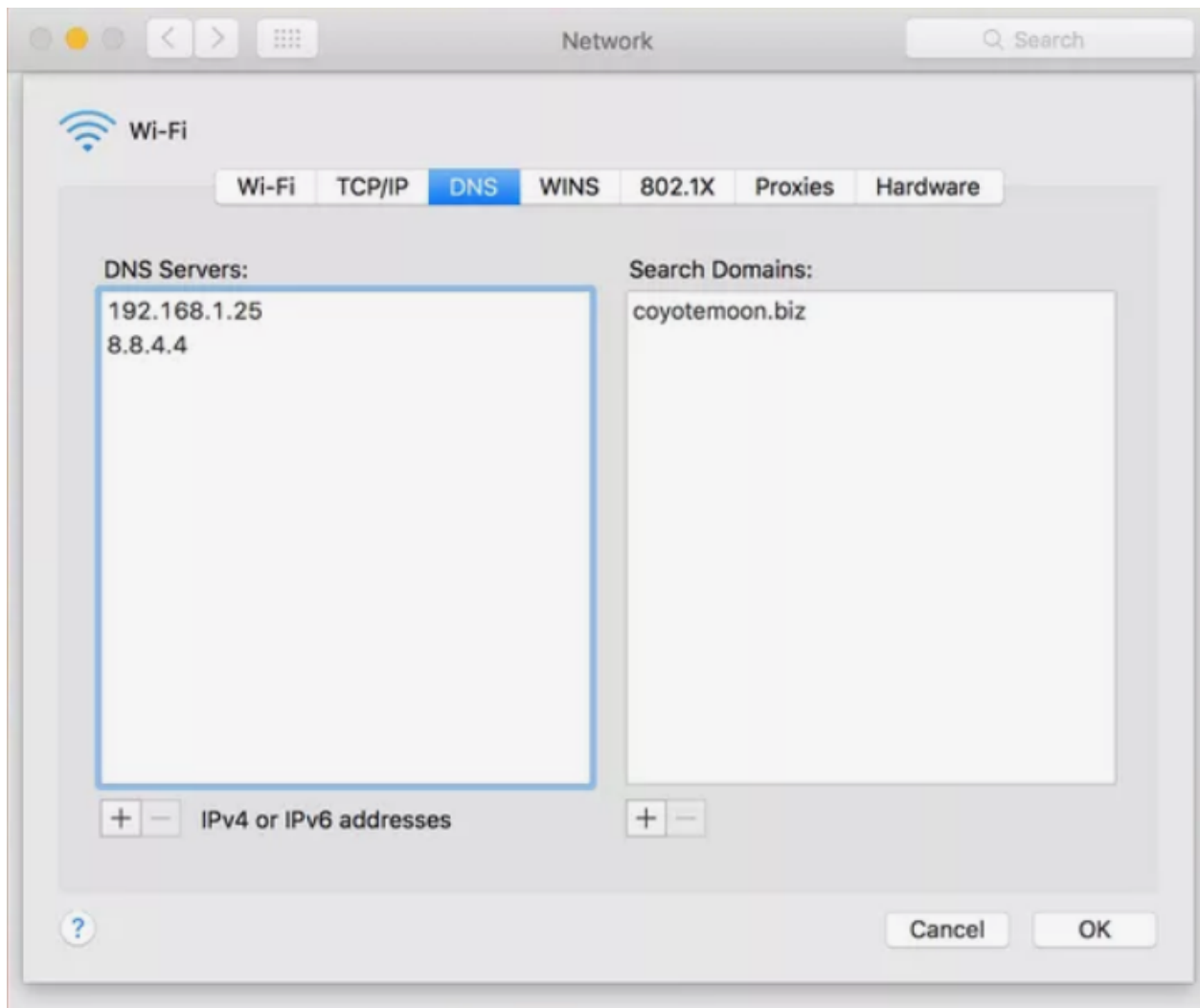


Fig. 10: MacOS: network settings

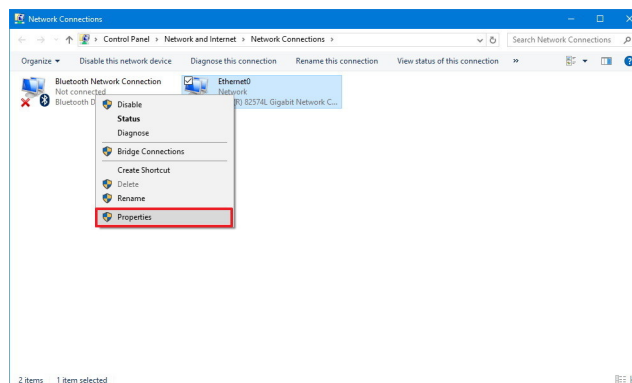


Fig. 11: Windows: network connections

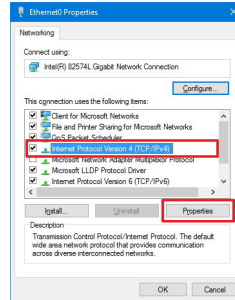


Fig. 12: Windows: ethernet properties

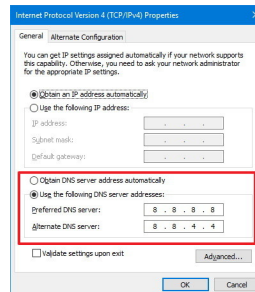


Fig. 13: Windows: internet protocol properties

- *Assumption*
- *Step by step*

### 87.6.1 Assumption

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	<a href="http://project-1.loc">http://project-1.loc</a>	project-1.loc
www.project-1	loc	<a href="http://www.project-1.loc">http://www.project-1.loc</a>	www.project-1.loc

#### Step by step

When using Docker on Linux you can use 127.0.0.1 for the IP address.

1. Open `/etc/hosts` with root privileges or via `sudo` with your favorite editor

```
host> sudo vi /etc/hosts
```

2. Add DNS records for the above listed examples:

Listing 5: /etc/hosts

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

### 3. Save the file and verify the DNS entries with the ping command

```
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
```

## 87.7 Add project hosts entry on MacOS

On MacOS, custom DNS entries can be added to the `/etc/hosts` and will take precedence over the same entries provided by any DNS server.

### Table of Contents

- *Assumption*
  - *Docker for Mac*
  - *Docker Toolbox*

### 87.7.1 Assumption

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	<a href="http://project-1.loc">http://project-1.loc</a>	project-1.loc
www.project-1	loc	<a href="http://www.project-1.loc">http://www.project-1.loc</a>	www.project-1.loc

### Docker for Mac

When using Docker for Mac you can use `127.0.0.1` for the IP address.

1. Open `/etc/hosts` with administrative privileges or via `sudo` with your favorite editor

```
host> sudo vi /etc/hosts
```

2. Add DNS records for the above listed examples:

Listing 6: /etc/hosts

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

3. Save the file and verify the DNS entries with the `ping` command

```
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
```

## Docker Toolbox

When using the Docker Toolbox, you cannot use `127.0.0.1` for DNS entries, but rather need to use the IP address of the Docker Toolbox machine instead.

**See also:**

*[Find Docker Toolbox IP address](#)*

For this example we will assume the Docker Toolbox IP address is `192.168.99.100`.

1. Open `/etc/hosts` with administrative privileges or via `sudo` with your favorite editor

```
host> sudo vi /etc/hosts
```

2. Add DNS records for the above listed examples:

Listing 7: /etc/hosts

```
192.168.99.100 project-1.loc
192.168.99.100 www.project-1.loc
```

3. Save the file and verify the DNS entries with the `ping` command

```
host> ping -c1 project-1.loc

PING project-1.loc (192.168.99.100) 56(84) bytes of data.
64 bytes from localhost (192.168.99.100): icmp_seq=1 ttl=64 time=0.066 ms
```

```
host> ping -c1 www.project-1.loc

PING www.project-1.loc (192.168.99.100) 56(84) bytes of data.
64 bytes from localhost (192.168.99.100): icmp_seq=1 ttl=64 time=0.066 ms
```

## 87.8 Add project hosts entry on Windows

On Windows, custom DNS entries can be added to the `C:\Windows\System32\drivers\etc` and will take precedence over the same entries provided by any DNS server.

## Table of Contents

- *Assumption*
  - *Docker for Windows*
  - *Docker Toolbox*

### 87.8.1 Assumption

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	<a href="http://project-1.loc">http://project-1.loc</a>	project-1.loc
www.project-1	loc	<a href="http://www.project-1.loc">http://www.project-1.loc</a>	www.project-1.loc

#### Docker for Windows

When using Docker for Windows you can use 127.0.0.1 for the IP address.

1. Open C:\Windows\System32\drivers\etc with administrative privileges via notepad.exe or any other text editor.
2. Add DNS records for the above listed examples:

Listing 8: C:WindowsSystem32driversetc

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

3. Save the file and verify the DNS entries with the ping command

```
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
```

#### Docker Toolbox

When using the Docker Toolbox, you cannot use 127.0.0.1 for DNS entries, but rather need to use the IP address of the Docker Toolbox machine instead.

**See also:**

*Find Docker Toolbox IP address*

For this example we will assume the Docker Toolbox IP address is 192.168.99.100.



1. Open `C:\Windows\System32\drivers\etc` with administrative privileges via `notepad.exe` or any other text editor.
2. Add DNS records for the above listed examples:

Listing 9: C:WindowsSystem32driversetc

```
192.168.99.100 project-1.loc
192.168.99.100 www.project-1.loc
```

3. Save the file and verify the DNS entries with the `ping` command

```
host> ping -c1 project-1.loc

PING project-1.loc (192.168.99.100) 56(84) bytes of data.
64 bytes from localhost (192.168.99.100): icmp_seq=1 ttl=64 time=0.066 ms
```

```
host> ping -c1 www.project-1.loc

PING www.project-1.loc (192.168.99.100) 56(84) bytes of data.
64 bytes from localhost (192.168.99.100): icmp_seq=1 ttl=64 time=0.066 ms
```

## 87.9 Find your user id and group id on MacOS

### Table of Contents

- *Docker for Mac vs Docker Toolbox*
  - *Docker for Mac*
  - *Docker Toolbox*
- *Find uid and gid*
  - *Find your user id (uid)*
  - *Find your group id (gid)*

### 87.9.1 Docker for Mac vs Docker Toolbox

#### Docker for Mac

On Docker for Mac (native Docker) you can open up any terminal you prefer, there are no other requirements.

#### Docker Toolbox

On Docker Toolbox it is important that you open up a Docker environment prepared terminal window.

See also:

- *Open a terminal on MacOS*

## 87.9.2 Find uid and gid

Open the correct terminal as described above and type the following commands:

### Find your user id (uid)

```
host> id -u
```

### Find your group id (gid)

```
host> id -g
```

## 87.10 Find your user id and group id on Windows

### Table of Contents

- *Docker for Windows*
- *Docker Toolbox*
  - *Find your user id (uid)*
  - *Find your group id (gid)*

### 87.10.1 Docker for Windows

On Docker for Windows it is **not necessary** to change uid and gid in your `.env` file.

---

**Note:** Docker for Windows is internally using network shares (SMB) to mount Docker volumes. This does not require to synchronize file and directory permissions via uid and gid.

---

### 87.10.2 Docker Toolbox

On Docker Toolbox it is important that you open up a Docker environment prepared terminal window.

#### See also:

- *Open a terminal on Windows*

### Find your user id (uid)

Type the following command to retrieve the correct uid.

```
host> id -u
```

## Find your group id (gid)

Type the following command to retrieve the correct gid.

```
host> id -g
```

## 87.11 Find Docker and Docker Compose version

Open a terminal and type the following:

```
# Get Docker version
host> docker --version

# Get Docker Compose version
host> docker-compose --version
```

See also:

- [Open a terminal on MacOS](#)
- [Open a terminal on Windows](#)

## 87.12 Move projects to a different directory

No matter if your projects are already in a different location or if you want to move them out of the Devilbox git directory now, you can do that in a few simple steps.

### Table of Contents

- [Projects in an absolute path](#)
- [Projects adjacent to Devilbox directory](#)

### 87.12.1 Projects in an absolute path

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 10: .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/`.

## 87.12.2 Projects adjacent to Devilbox directory

Consider the following directory setup:

```
|
+- devilbox/
|
+- projects/
  |
  + project1/
  | |
  | + htdocs/
  |
  + project2/
  |
  + htdocs/
```

Independently of where the Devilbox directory is located, you can achieve this structure via relative path settings.

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 11: .env

```
HOST_PATH_HTTPD_DATADIR=../projects
```

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

## 87.13 Host address alias on MacOS

In order for Xdebug to work on Docker for MacOS, the container needs a well known IP address for its Xdebug remote host. This is achieved by adding an alias to the loopback device.

### Table of Contents

- *One-time alias*
- *Boot persistent alias*

### 87.13.1 One-time alias

In order to create this alias for testing purposes, which does not survive reboots, you can issue the command manually with `sudo` or root privileges.

```
host> sudo ifconfig lo0 alias 10.254.254.254
```

### 87.13.2 Boot persistent alias

If you want to have this alias persistent across reboot, you need to download and enable a `plist` file:

```
# Download the plist into the correct location
host> sudo curl -o \
    /Library/LaunchDaemons/org.devilbox.docker_10254_alias.plist \
    https://raw.githubusercontent.com/devilbox/xdebug/master/osx/org.devilbox.
↳ docker_10254_alias.plist

# Enable without reboot
host> sudo launchctl load /Library/LaunchDaemons/org.devilbox.docker_10254_alias.plist
```

See also:

- *Configure PHP Xdebug*
- 
- 

## 87.14 Docker Toolbox and the Devilbox

Docker Toolbox is a legacy solution to bring Docker to systems which don't natively support Docker. This is achieved by starting a virtualized Linux instance (e.g.: inside VirtualBox) and have Docker run inside this machine.

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).

However, there are a few stumbling blocks you need to pay attention to in order to use the Devilbox at its full potential.

See also:

#### Docker Toolbox

- 
- 
- 
- 

#### Table of Contents

- *Devilbox listening address configuration*
- *Find the Docker Toolbox IP address*
- *Project DNS record pitfalls*

- *Auto-DNS via port forwarding*
  - *How does Auto-DNS work?*
  - *How to fix it for Docker Toolbox*
- *Mount shared folders*
  - *MacOS*
  - *Windows*

### 87.14.1 Devilbox listening address configuration

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 the virtual machine (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=
```

---

**Important:** The variable should exist, but there should not be any value after the equal sign.

---

**See also:**

*.env file*

### 87.14.2 Find the Docker Toolbox IP address

The Devilbox intranet will not be available under `127.0.0.1` or `localhost` as it does not run on your host operating system, but on a virtualized Linux machine which has a different IP address.

To find out the IP address on which Docker Toolbox is running you have to use the `docker-machine` command. Open a terminal and type the following:

```
host> docker-machine ip default  
192.168.99.100
```

The above example outputs `192.168.99.100`, but this might be different on your machine.

In this example I would then paste `http://192.168.99.100` in the web browsers address bar to reach the Devilbox intranet.

**See also:**

- *Open a terminal on MacOS*
- *Open a terminal on Windows*
- *Find Docker Toolbox IP address*

### 87.14.3 Project DNS record pitfalls

When creating manual DNS records per project, you have to keep in mind that you cannot use `127.0.0.1` for the IP address part. You have to use the IP address of the Docker Toolbox virtual machine as was shown in the above example.

Assuming the Docker Toolbox IP address is: `192.168.99.100`, you have to create DNS records as follows:

Listing 12: `/etc/resolv.conf` or `C:\Windows\System32\drivers\etc`

```
192.168.99.100 project.loc
```

See also:

- [Add project hosts entry on MacOS](#)
- [Add project hosts entry on Windows](#)
- [Find Docker Toolbox IP address](#)

### 87.14.4 Auto-DNS via port forwarding

In order to make Auto-DNS for projects work as it does for native Docker implementations you will have to do some prior configuration.

#### How does Auto-DNS work?

Auto-DNS is a catch-all DNS resolver for your chosen *TLD\_SUFFIX* that will redirect any domain to `127.0.0.1`. Unfortunately Docker Toolbox does not listen on that IP address.

#### How to fix it for Docker Toolbox

To overcome this problem, you will have to create three port forwards on your host operating system from the Docker machine IP address for DNS (port 53), `http` (port 80) and `https` (port 443) to `127.0.0.1` on your host os.

Assuming the Docker Toolbox IP address is `192.168.99.100` the three port forwards must be as follows:

From IP	From port	To IP	To port
192.168.99.100	53	127.0.0.1	53
192.168.99.100	80	127.0.0.1	80
192.168.99.100	443	127.0.0.1	443

See also:

- [Find Docker Toolbox IP address](#)
- [SSH port-forward on Docker Toolbox from host](#)
- [Setup Auto DNS](#)

### 87.14.5 Mount shared folders

Docker Toolbox will automatically set up a shared directory between your host operating system and the virtual Linux machine. Only files and directories within this shared directory can be used to be mounted into Docker container. If

you plan to mount files or directories outside of this default path you have to create a new shared directory as described below.

## MacOS

When you want to have your projects reside not somewhere in the `/Users` directory, ensure you have read, understood and applied the following:

“By default, Toolbox only has access to the `/Users` directory and mounts it into the VMs at `/Users`. If your project lives elsewhere or needs access to other directories on the host filesystem, you can add them.”

**See also:**

## Windows

When you want to have your projects reside not somewhere in the `C:\Users` directory, ensure you have read, understood and applied the following:

“By default, Toolbox only has access to the `C:\Users` directory and mounts it into the VMs at `/c/Users`. If your project lives elsewhere or needs access to other directories on the host filesystem, you can add them, using the VirtualBox UI.”

**See also:**

## 87.15 Find Docker Toolbox IP address

### Table of Contents

- [\*Get IP address\*](#)
- [\*What to do with it\*](#)

### 87.15.1 Get IP address

1. Open an environment prepared Terminal
2. Enter the following command to get the IP address of the Docker Toolbox virtual machine:

```
host> docker-machine ip default  
  
192.168.99.100
```

The above example outputs `192.168.99.100`, but this might be a different IP address on your machine.

**See also:**

- [\*Open a terminal on MacOS\*](#)
- [\*Open a terminal on Windows\*](#)



## 87.15.2 What to do with it

The Docker Toolbox IP address is the address where the Devilbox intranet as well as all of its projects will be available at.

- Use it to access the intranet via your browser (`http://192.168.99.100` in this example)
- Use it for manual DNS entries

See also:

- *Add project hosts entry on MacOS*
- *Add project hosts entry on Windows*

## 87.16 SSH into Docker Toolbox

### Table of Contents

- *Requirements*
- *Manual*
  - *Gather all information*
  - *Gather specific information*
  - *SSH into Docker Toolbox*
- *Automated*

### 87.16.1 Requirements

You shell must have an SSH client (the `ssh` command or equivalent).

See also:

- *Open a terminal on MacOS*
- *Open a terminal on Windows*
- *Find Docker Toolbox IP address*

### 87.16.2 Manual

Before going to use the automated approach, you should understand how to fetch all required information via the `docker-machine` command.

#### Gather all information

1. Get active Toolbox machine name

```
host> docker-machine active
default
```

## 2. Print all information

```
host> docker-machine -D ssh default
Host : localhost
Port : 51701
User : docker
Key : .docker\machine\machines\default\id_rsa
```

## Gather specific information

### 1. Get active Toolbox machine name

```
host> docker-machine active
default
```

### 2. Get SSH username (Using machine name default from above)

```
host> docker-machine inspect default --format={{.Driver.SSHUser}}
docker
```

### 3. Get SSH public key (Using machine name default from above)

```
host> docker-machine inspect default --format={{.Driver.SSHKeyPath}}
.docker\machine\machines\default\id_rsa
```

### 4. Get local SSH port (Using machine name default from above)

```
host> docker-machine inspect default --format={{.Driver.SSHPort}}
51701
```

### 5. Get Docker Toolbox IP address (Using machine name default from above)

```
host> docker-machine ip default
192.168.99.100
```

## SSH into Docker Toolbox

Now with the above gathered information you can ssh into Docker Toolbox in two different ways:

#### 1. via local port-forwarded ssh port (automatically forwarded by Docker Toolbox)

```
host> ssh -i .docker\machine\machines\default\id_rsa -p 51701 docker@127.0.0.1
```

#### 2. via Docker Toolbox IP address

```
host> ssh -i .docker\machine\machines\default\id_rsa docker@192.168.99.100
```

## 87.16.3 Automated

Instead of typing all of the above manually each time, you can also create a small bash script to automate this.

#### 1. Create a file `ssh-docker.sh` and add the following to it:

Listing 13: ssh-docker.sh

```
#!/bin/bash
docker_machine_name=$(docker-machine active)
docker_ssh_user=$(docker-machine inspect $docker_machine_name --format={{.Driver.
↪SSHUser}})
docker_ssh_key=$(docker-machine inspect $docker_machine_name --format={{.Driver.
↪SSHKeyPath}})
docker_ssh_port=$(docker-machine inspect $docker_machine_name --format={{.Driver.
↪SSHPort}})

ssh -i $docker_ssh_key -p $docker_ssh_port $docker_ssh_user@localhost
```

2. Run it:

```
host> bash ssh-docker.sh
```

See also:

## 87.17 SSH port-forward on Docker Toolbox from host

**Note:** This is a **Local SSH port-forward** (`ssh -L`)

### Table of Contents

- *Requirements*
- *Overview*
  - *General command*
  - *Command example*
- *Examples*
  - *Make host-based MySQL available on Docker Toolbox*
  - *Make host-based PostgreSQL available on Docker Toolbox*

### 87.17.1 Requirements

You **host operating system** must have an **SSH server** installed, up and running.

See also:

- *Open a terminal on MacOS*
- *Open a terminal on Windows*
- *Find Docker Toolbox IP address*
- *SSH into Docker Toolbox*
-

## 87.17.2 Overview

This is a **local** SSH port-forward (`ssh -L`). In other words, the Docker Toolbox machine will make a port **locally available** from somewhere else. Therefore the process must be initiated on the Docker Toolbox machine.

### General command

The following represents the general structure of a local ssh port-forward:

```
ssh -L <DockerToolbox_Port>:<HostOS_SRV_IP>:<HostOS_SRV_Port> <HostOS_SSH_USER>@  
↪<HostOS_SSH_IP>
```

<DockerToolbox_Port>	The port on the Docker Toolbox machine the service should be made available
<HostOS_SRV_IP>	The IP address on the host os, where the service is currently listening
<HostOS_SRV_PORT>	The port on the host os, where the service is bound to
<HostOS_SSH_USER>	The username of the host os SSH server for the connection
<HostOS_SSH_IP>	The IP address of the host at which the SSH server is reachable

### Command example

Making `127.0.0.1:10000` from host os available on `0.0.0.0:8080` on Docker Toolbox machine:

```
ssh -L 8080:127.0.0.1:10000 user@172.16.0.1
```

8080	Docker Toolbox should make the port available on itself on this port
127.0.0.1	The service currently listens on that IP address on the host os
10000	The service is currently bound to that port on the host os
user	The username of the host os SSH server for the connection
172.16.0.1	The IP address of the host at which the SSH server is reachable

## 87.17.3 Examples

For this example we assume the following information:

- Docker Toolbox IP address is `192.168.99.100`
- Host os IP address where SSH server is listening is `172.16.0.1`
- Host SSH username is `user`

### Make host-based MySQL available on Docker Toolbox

1. Gather the IP address on your host os where the SSH server is listening
2. SSH into the Docker Toolbox machine
3. Forward: `127.0.0.1:3306` from host os to `0.0.0.0:3306` on Docker Toolbox

```
toolbox> ssh -L 3306:127.0.0.1:3306 user@172.16.0.1
```

## Make host-based PgSQL available on Docker Toolbox

1. Gather the IP address on your host os where the SSH server is listening
2. SSH into the Docker Toolbox machine
3. Forward: 127.0.0.1:5432 from host os to 0.0.0.0:5432 on Docker Toolbox

```
toolbox> ssh -L 5432:127.0.0.1:5432 user@172.16.0.1
```

## 87.18 SSH port-forward on host to Docker Toolbox

**Note:** This is a **Remote SSH port-forward** (`ssh -R`)

### Table of Contents

- *Requirements*
- *Overview*
  - *General command*
  - *Command example*
- *Examples*
  - *Make host-based MySQL available on Docker Toolbox*
  - *Make host-based PgSQL available on Docker Toolbox*

### 87.18.1 Requirements

You shell must have an **SSH client** (the `ssh` command or equivalent).

**See also:**

- *Open a terminal on MacOS*
- *Open a terminal on Windows*
- *Find Docker Toolbox IP address*
- *SSH into Docker Toolbox*
- 

### 87.18.2 Overview

This is a **remote** SSH port-forward (`ssh -R`). In other words, the host os will make the port **remotely availabl** on the Docker Toolbox machine. Therefore the process must be initiated on the host os.

## General command

The following represents the general structure of a remote ssh port-forward:

```
ssh -R <DockerToolbox_Port>:<HostOS_SRV_IP>:<HostOS_SRV_Port> <DockerToolbox_SSH_USER>  
↪@<DockerToolbox_SSH_IP>
```

<DockerToolbox_Port>	The port on the Docker Toolbox machine the service should be made available
<HostOS_SRV_IP>	The IP address on the host os, where the service is currently listening
<HostOS_SRV_PORT>	The port on the host os, where the service is bound to
<DockerToolbox_SSH_USER>	The username of the host os SSH server for the connection
<DockerToolbox_SSH_IP>	The IP address of the host at which the SSH server is reachable

## Command example

Making 127.0.0.1:10000 from host os available on 0.0.0.0:8080 on Docker Toolbox machine:

```
ssh -R 8080:127.0.0.1:10000 docker@192.168.99.100
```

8080	Docker Toolbox should make the port available on itself on this port
127.0.0.1	The service currently listens on that IP address on the host os
10000	The service is currently bound to that port on the host os
docker	The username of the Docker Toolbox SSH server for the connection
192.168.99.100	The IP address of the Docker Toolbox at which the SSH server is reachable

## 87.18.3 Examples

For this example we assume the following information:

- Docker Toolbox IP address is 192.168.99.100
- Docker Toolbox SSH username is `docker`

### Make host-based MySQL available on Docker Toolbox

1. Open a terminal on your host os
2. Forward: 127.0.0.1:3306 from host os to 0.0.0.0:3306 on Docker Toolbox

```
toolbox> ssh -R 3306:127.0.0.1:3306 docker@192.168.99.100
```

### Make host-based PgSQL available on Docker Toolbox

1. Open a terminal on your host os
2. Forward: 127.0.0.1:5432 from host os to 0.0.0.0:5432 on Docker Toolbox

```
toolbox> ssh -R 5432:127.0.0.1:5432 docker@192.168.99.100
```

## 87.19 Open a terminal on MacOS

See also:

*Open a terminal on Windows*

### Table of Contents

- *Docker for Mac*
- *Docker Toolbox*
  - *Via Launcher*
  - *Different terminal*

### 87.19.1 Docker for Mac

Docker for Mac (the native Docker implementation) does not have any special requirements for initial environment variable setup. Simply open your terminal of choice from the **Launchpad** (`Terminal.app` or `iTerm.app`).

See also:

### 87.19.2 Docker Toolbox

Docker Toolbox provides a launcher to open an environment prepared terminal, but you can also do it manually with a terminal of your choice.

#### Via Launcher

1. Open the **Launchpad** and locate the Docker Quickstart Terminal icon.
2. Click the icon to launch a Docker Quickstart Terminal window.

The terminal does a number of things to set up Docker Quickstart Terminal for you.

```
Last login: Sat Jul 11 20:09:45 on ttys002
bash '/Applications/Docker Quickstart Terminal.app/Contents/Resources/Scripts/start.sh'
↪ '
Get http://var/run/docker.sock/v1.19/images/json?all=1&filters=%7B%22dangling%22%3A
↪ %5B%22true%22%5D%7D: dial unix /var/run/docker.sock: no such file or directory. Are
↪ you trying to connect to a TLS-enabled daemon without TLS?
Get http://var/run/docker.sock/v1.19/images/json?all=1: dial unix /var/run/docker.
↪ sock: no such file or directory. Are you trying to connect to a TLS-enabled daemon
↪ without TLS?
-bash: lolcat: command not found

mary at meepers in ~
$ bash '/Applications/Docker Quickstart Terminal.app/Contents/Resources/Scripts/start.
↪ sh'
Creating Machine dev...
Creating VirtualBox VM...
Creating SSH key...
Starting VirtualBox VM...
```

(continues on next page)

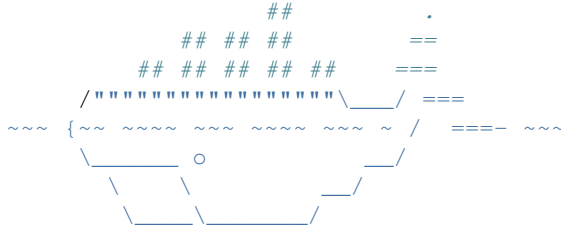


Fig. 14: Copyright docs.docker.com



(continued from previous page)

```
Starting VM...
To see how to connect Docker to this machine, run: docker-machine env dev
Starting machine dev...
Setting environment variables for machine dev...
```



The Docker Quick Start Terminal is configured to use Docker with the "default" VM.

You can now use this terminal window to apply all your Docker and Devilbox related commands.

### Different terminal

If you rather want to use a different terminal, you can accomplish the same behaviour.

1. Open your terminal of choice
2. Type the following to prepare environment variables

```
$ (docker-machine env default)
```

You can now use this terminal window to apply all your Docker and Devilbox related commands.

See also:

## 87.20 Open a terminal on Windows

See also:

*Open a terminal on MacOS*

### Table of Contents

- *Docker for Windows*
- *Docker Toolbox*

### 87.20.1 Docker for Windows

Docker for Windows (the native Docker implementation) does not have any special requirements for initial environment variable setup. Simply open either

- Command Prompt
- PowerShell

---

**Important:** Do not open **PowerShell ISE**

---

See also:

## 87.20.2 Docker Toolbox

1. On your Desktop, find the Docker QuickStart Terminal icon.

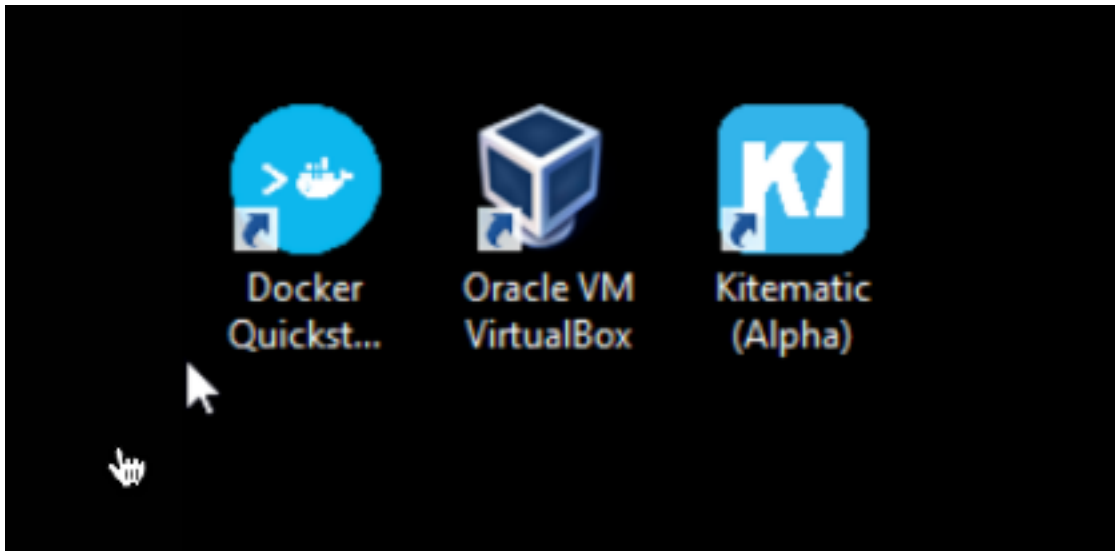
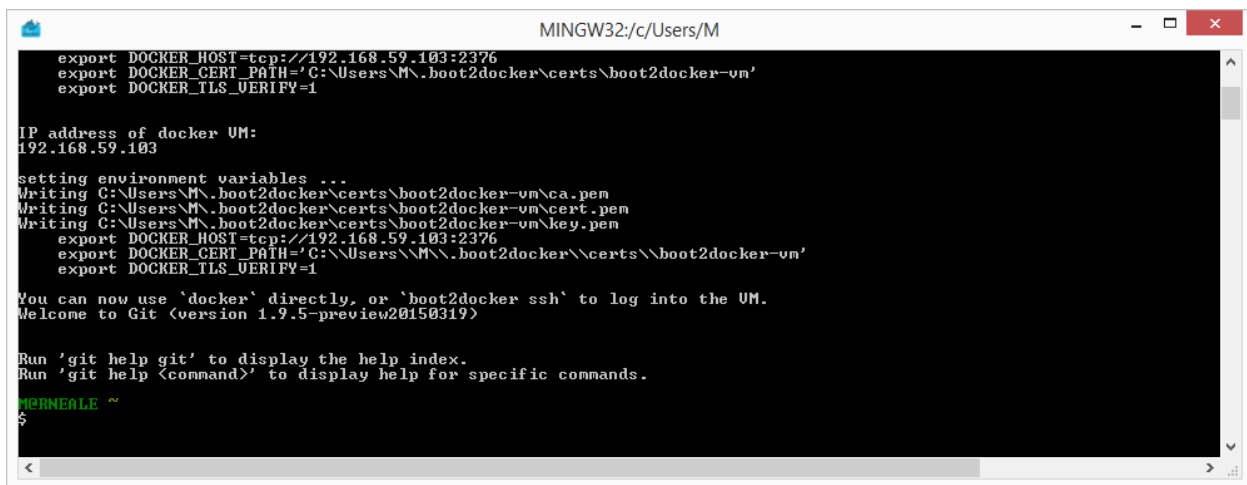


Fig. 15: Copyright docs.docker.com

2. Click the Docker QuickStart icon to launch a pre-configured Docker Toolbox terminal.

If the system displays a **User Account Control** prompt to allow VirtualBox to make changes to your computer. Choose **Yes**.

A screenshot of a Windows terminal window titled 'MINGW32:/c/Users/M'. The terminal displays the following text:

```
export DOCKER_HOST=tcp://192.168.59.103:2376
export DOCKER_CERT_PATH='C:\Users\M\boot2docker\certs\boot2docker-vm'
export DOCKER_TLS_VERIFY=1

IP address of docker VM:
192.168.59.103

setting environment variables ...
Writing C:\Users\M\boot2docker\certs\boot2docker-vm\ca.pem
Writing C:\Users\M\boot2docker\certs\boot2docker-vm\cert.pem
Writing C:\Users\M\boot2docker\certs\boot2docker-vm\key.pem
export DOCKER_HOST=tcp://192.168.59.103:2376
export DOCKER_CERT_PATH='C:\Users\M\boot2docker\certs\boot2docker-vm'
export DOCKER_TLS_VERIFY=1

You can now use 'docker' directly, or 'boot2docker ssh' to log into the VM.
Welcome to Git (version 1.9.5-preview20150319)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

MERNEALE ~
$
```

Fig. 16: Copyright docs.docker.com

The terminal runs a special bash environment instead of the standard Windows command prompt. The bash environment is required by Docker.

You can now use this terminal window to apply all your Docker and Devilbox related commands.

**See also:**



---

### Blogs, Videos and Use-cases

---

#### Table of Contents

- *Official videos*
- *Conferences*
  - *DrupalCamp Ghent 2018*
- *Blog posts*
- *Use-cases*
  - *Joomla's Continuous Integration*
- *Add your story*

### 88.1 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.

### 88.2 Conferences

#### 88.2.1 DrupalCamp Ghent 2018

Simple local development with Devilbox:

- 
-



## 88.3 Blog posts

The following shows a list of blogs that give a nice and objective introduction to the Devilbox.

Title	Language
	English
	German

## 88.4 Use-cases

### 88.4.1 Joomla's Continuous Integration

Joomla has created a as their project using a modified version of the Devilbox.

## 88.5 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>