

Bachelor's thesis

Degree Programme in Information Technology

Information Technology

2017

Rubén Cayetano Díaz Alonso

SOFTWARE CONTAINERIZATION WITH DOCKER

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology

2017 | 47

Rubén Cayetano Díaz Alonso

SOFTWARE CONTAINERIZATION WITH DOCKER

The main purpose of this thesis was to introduce software containerization, a type of OS level virtualization which has become very popular among organizations and individuals, and its most used implementation, Docker.

Docker fulfills the needs of developers who need to provide their software to other users in an environment where all the required dependencies and settings are already present, giving every user the possibility to run the application in the same environment.

Docker also implements a minimum level of isolation (process sandboxing) from other processes being run in the machine hosting the Docker daemon using Linux's kernel features. The overhead of Docker containers is, in most cases, negligible, and it can be an alternative, under some circumstances, to traditional virtual machines.

While software containerization is not new and in fact there are multiple implementations of this technology in the different operating systems available in the market, Docker as a platform has popularized containerization.

The learning curve of Docker is not steep, which translates into a handful of vendors offering resources for the different users of Docker and a considerable user base.

This thesis shows the most important concepts of containerization and what advantages Docker as an implementation offers in addition to the technology, how it compares with virtual machines, as well as instructions for setting it up and using it including two practical cases: a public-facing Python 3 application and a WordPress LAMP stack using Docker Compose.

KEYWORDS:

Docker, containerization, software, engineering, virtualization, Linux

CONTENTS

CONTENTS	8
APPENDICES	8
FIGURES	9
LIST OF ABBREVIATIONS (OR) SYMBOLS	10
1 INTRODUCTION	1
2 DOCKER OVERVIEW	3
2.1 Docker containers vs. Virtual Machines	3
2.2 Separation between client and daemon	5
2.3 Docker image system	6
3 DOCKER IN PRACTICE	8
3.1 Setting up Docker	8
4 Windows 10	8
5 macOS	11
6 Linux (Ubuntu 16.04)	15
6.1 Creating a Docker container image	17
6.2 Use case: WordPress	22
7 CONCLUSION	25
REFERENCES	27

APPENDICES

- Appendix 1. Contents of the Python application files
- Appendix 2. Docker build output for Python application
- Appendix 3. Contents of the docker-compose.yaml
- Appendix 4. Output of Docker Compose

FIGURES

Figure 1. Official Docker logo (Docker Inc, "Marks and Logos", 2017).	3
Figure 2. Architectural view of VM vs Docker (Docker Inc, "What is Docker", 2017).	4
Figure 3. Docker Engine parts representation (Docker Inc, "Understanding Docker", 2017).	6
Figure 4. Structure of a container's image layers (Docker Inc, "Understand images, containers, and storage drivers", 2017).	7
Figure 5. Docker download page (platform selection, Windows selected).	9
Figure 6. Docker installation assistant (Windows).	9
Figure 7. Docker asking to enable Hyper-V feature.	10
Figure 8. Docker asking to run Docker service.	10
Figure 9. Running <i>hello-world</i> container (Windows).	11
Figure 10. Docker download page (platform selection, Mac selected).	12
Figure 11. Installing Docker in macOS.	12
Figure 12. macOS security warning about newly downloaded Docker.	13
Figure 13. Docker welcome popup in macOS.	13
Figure 14. Docker asking for privileged access (macOS).	14
Figure 15. Running hello-world container (macOS).	15
Figure 16. Running hello-world container (Ubuntu).	17
Figure 17. Container image representation (Singh A, 2015).	18
Figure 18. Output of command 9.	20
Figure 19. Output of command 10.	20
Figure 20. Output of command 11.	20
Figure 21. Index page of <i>python_app</i> .	21
Figure 22. Example result of <i>python_app</i> .	21
Figure 23. WordPress installation webpage.	24

LIST OF ABBREVIATIONS (OR) SYMBOLS

API	Application programming interface
APT	Advanced Packaging Tool
ASCII	American Standard Code for Information Interchange
aufs	Advanced Multi-Layered Unification Filesystem
DMG	Apple Disk Image
GPG	GNU Privacy Guard
HTTP	Hypertext Transfer Protocol
LTS	Long Term Support
MSI	Microsoft Installer
OS	Operating System
PHP	PHP: Hypertext Preprocessor
REST	Representational state transfer
UUID	Universally unique identifier
VM	Virtual Machine

1 INTRODUCTION

The needs for virtualization date from the 1960s when the Massachusetts Institute of Technology started its Project MAC, a research project funded by the Defense Advanced Research Projects Agency government agency with ambitious goals. Computers could only perform one task at a time due to technical limitations but the project required to run more than that.

New hardware/software was requested from different vendors that could support time-sharing – this is, multitasking. *IBM CP-40* (the precursor of the commercial *IBM CP-67*) is considered the first Main Frame that supported virtualization under one of the IBM's OS (Operating System) known as *IBM System/360 Model 67*. IBM then met MIT's needs (Conroy 2011).

Virtualization and OS in general have been constantly evolving since then. Nowadays most of the computer power users do not think of virtualization as a way of running more than one single simple task at once but more as a way of running different software (such as operating systems) within the same machine without interfering the host OS.

There is not a single way to virtualize software. There is not even a right way to do it. Every user – be individual or organization – has completely different needs. Nowadays there is a wide range of solutions that fit in the different situations where more or less isolation is required.

The typical virtualization method among individuals and organizations until recently has been using virtual machines provided by *hypervisors* from different vendors such as Oracle (VirtualBox), VMWare (vSphere) or Microsoft (Hyper-V) among others (Kleyman 2012). The main advantage of full virtualization is that the isolation is at the highest level from the host machine (the machine supporting the virtualization) which implies the possibility of having a wider selection of supported OS and a clear separation between the running host OS and the virtualized one, providing a safer environment. Nevertheless, virtual machines have a main drawback: the amount of resources needed to virtualize is higher as it is a completely isolated system and hence does not share anything with the host machine, while containers share the host OS kernel (Docker Inc, "Docker overview", 2016).

Software containers offer a different type of virtualization. The advantage of containers is that they offer a reduced isolation level (compared to Virtual Machines) at an affordable cost since it uses the already running host OS kernel. Docker, as an open-source platform, provides abstraction layers that help the user to use this type of virtualization. Due to its design, it offers a convenient way for users to create new containers and make applications portable so that other users do not need to deal with dependencies (also known as “dependency hell”) but at the same time, they do not need to have a virtual machine running the application with all the associated costs (Felter ym. 2014).

Docker has evolved considerably in the recent years. The fact that all the products are open-source made the community to play a huge role, weighting what should be developed next. The user base keeps growing and many large companies have publicly announced that they are actively using Docker as part of their own platform (Docker Inc, “Docker Customers”, 2017).

However, this rapid evolution makes Docker a product that cannot still be considered stable and reliable (compared to traditional Virtual Machines) due to the fast pace of breaking changes. This fact, while it lasts, will push away users looking to enhance their critical infrastructure as the risks may be too high (Williams & Jackson, 2016).

While Docker is not a fit for every type of environment, users who do not need the full isolation provided by VMs (Virtual Machines), are aware of possible breaking changes in the short term and do not need to run an OS different than the one in the host machine may find Docker an option to be considered to achieve reproducible software environments - to move from the developer’s environment to the final production environment without unexpected issues on the way.

This thesis shows the most important concepts of containerization and what Docker offers in addition to the technology, how it compares with virtual machines, as well as instructions for setting it up and using it with some practical cases to demonstrate that Docker is a technology that can be used for different purposes.

2 DOCKER OVERVIEW

Docker (official logo shown in Figure 1) is a platform with different components that abstract away some of the arduous tasks that users would need to otherwise endure when dealing with low-level OS virtualization. In official words, Docker thrives to “wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries” (Docker documentation, 2016), so that its environment is similar no matter where it is being run.



Figure 1. Official Docker logo (Docker Inc, “Marks and Logos”, 2017).

The possibility of having reproducible software environments, which means that software behaves exactly the same in spite of running under different hardware, configuration and perhaps even different OS, is an appealing option for developers since they are able to work with the software under the intended original conditions and most of the otherwise unexpected issues would not be present anymore. This also helps the operations side of software – if the software runs properly in the container, the team is able to ship it to the final destination and expect to behave the same way (Boettiger, 2014).

2.1 Docker containers vs. Virtual Machines

Containers are lightweight (resource-wise) by default. A single host can run multiple containers at the same time and, in general, more containers than otherwise VMs. This encourages the user to separate each component of the application in different containers and link them together with tools provided by the Docker platform (such as Docker Compose) instead of trying to put every component inside a single VM so that

hardware resources could be saved. Figure 2 shows the main architectural difference between VMs and Docker.

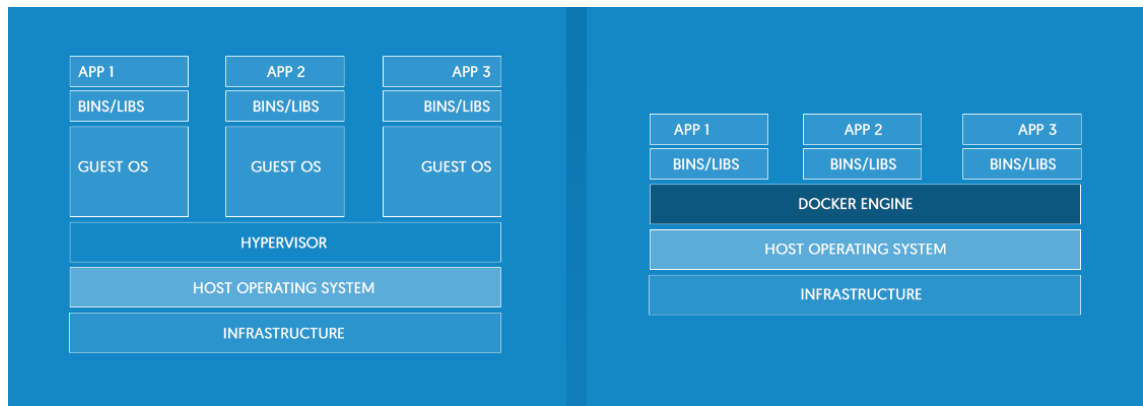


Figure 2. Architectural view of VM vs Docker (Docker Inc, “What is Docker”, 2017).

From the architectural point of view, VMs require that each application component has its own guest OS installed whereas Docker shares some of the resources from the host OS and therefore offers the following advantages:

- Small: the container requires the application data itself and the OS, but the kernel is shared with the host OS (Docker Inc, “Docker overview”, 2016).
- Fast: since there is no need to load the kernel, the container start process normally takes from milliseconds to few seconds. In practice, the virtualization is just an additional running process that is sandboxed from the rest, and creating new processes in the already running host OS is cheap (Seo ym. 2014).
- Resource usage flexibility: while VMs require strict limits regarding hardware usage, Docker also allows to set soft limits, which allows running containers to use underutilized resources that were allocated to other containers (Sharma ym. 2016)
- Enhanced reusability: it is simple to create containers based on other containers so that steps are not repeated. Moreover, different containers sharing the same base image can simultaneously run without the need of replicating the image in disk (Docker Inc, “Understand images, containers, and storage drivers”, 2017).

Docker also has drawbacks:

- Weaker isolation: there is a significant difference from the isolation that an hypervisor can offer. Software containers are sandboxed processes using some of Linux's kernel features (namespaces, cgroups, AppArmor, etc.). Due to this nature, this cannot be radically changed (Sharma ym. 2016)
- Limited OS selection: while a VM with an OS that is different from the host machine can be run, this is not possible with containers since the host OS kernel is shared. This means that a Linux host machine can only run other *NIX systems inside containers (Sharma ym. 2016)
- Reduced stability: the fast pace of breaking changes in the platform is a common side-effect in new technologies, and Docker is no exception (Williams & Jackson, 2016). Users looking for a mature and stable software to run a critical infrastructure that offers similar features should use traditional virtualization.
- Performance penalties: while Docker containers achieve nearly bare-metal performance in general, the usage of Docker NAT adds CPU overhead and the I/O speed of container storage is slower than bare-metal or data volumes attached to the container (Felter ym. 2014).

2.2 Separation between client and daemon

The core component of the Docker platform is called Docker Engine, which is fully open-source (Docker Inc, "Docker Engine", 2017). Docker Engine is composed of three well differentiated parts:

- A server, which is a daemon process (i.e., a program that is running for a long time serving a specific purpose) which manages the different types of Docker objects: containers, images, etc., known as *dockerd*.
- A REST (Representational state transfer) API (Application programming interface) that offers an intermediate layer to interact with the Docker server and control all its features.
- A command line interface which allows the user to communicate and interact with the server through the REST API.

This separation (graphically represented in Figure 3) allows the user to decide where to place the server. For example, a typical user places the Docker client in the same

machine as the Docker server. However, if the user's machine is a low-powered one (for example, a Raspberry Pi) or the software/OS is outdated and could not run the server, the user is still be able to install the server in some other machine and use the client in its own machine as long as it has access to the port where the server is running. There will be no difference in terms of usability – the user is able to execute any container and perform the same actions as if the server was running in its own machine (Miell & Hobson, 2016).

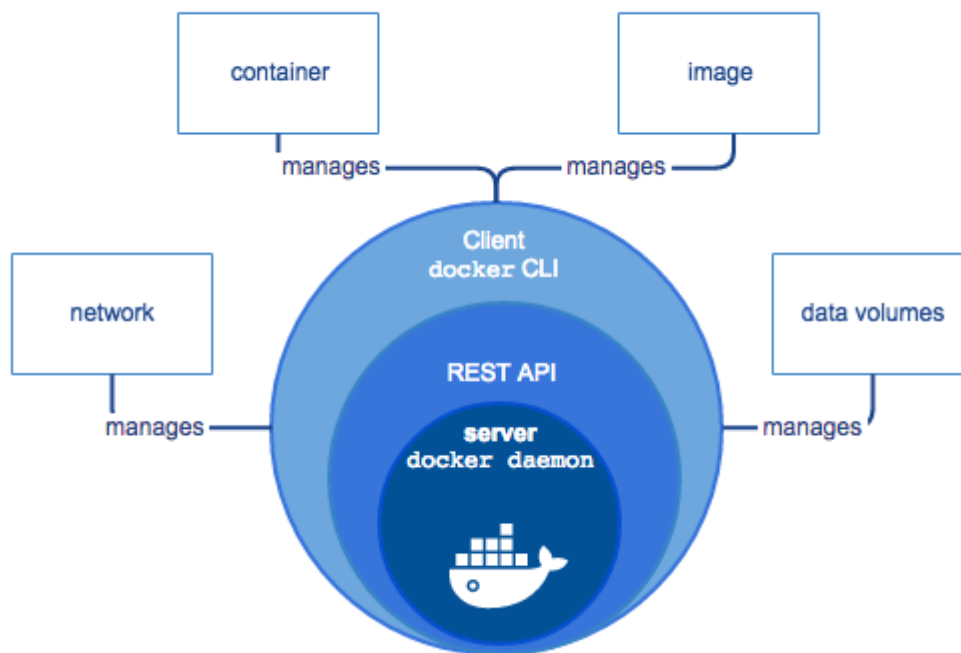


Figure 3. Docker Engine parts representation (Docker Inc, “Understanding Docker”, 2017).

2.3 Docker image system

A Docker container is composed of read-only image layers as shown in Figure 4. A layer represents the filesystem differences (for example: modified files, permissions, configurations, etc.) starting from a completely empty space. The Docker storage driver is then responsible for stacking all the layers together and form a functional container (Docker Inc, “Understand images, containers, and storage drivers”, 2017).

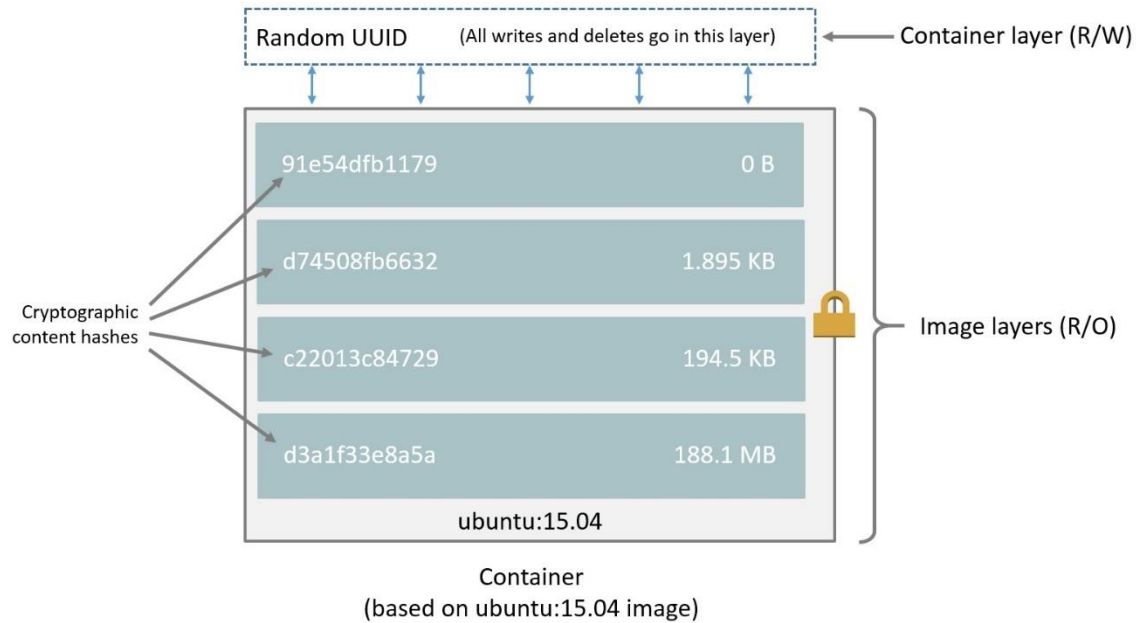


Figure 4. Structure of a container's image layers (Docker Inc, "Understand images, containers, and storage drivers", 2017).

Each of the layers has a cryptographic hash that identifies it, and could vary in size. The layers are read-only, i.e., all the changes done in a running container will be written in a new container layer on top of the rest, with a random UUID (Universally unique identifier). When the container is deleted, this layer will also be deleted but the underlying ones will exist.

This method of creating images makes convenient to share the same layers between different running containers while still having their own different data state (the container layer). For example, if ten different containers are using the same Ubuntu image pictured in Figure 4 and each has a different purpose (one is a web server, another one serves as database, etc.), they will all use the same underlying layers from the pictured base image without duplicating them while having a thin layer on top of these with the data used for its custom purpose Singh (2015).

3 DOCKER IN PRACTICE

This section explains how to set up Docker Engine in the different available modern and mainstream OS available in the market, the process of creation of a Docker container image to run a Python web application, and a typical use case using one of the official Docker tools named Docker Compose to spin up a set of containers in an organized way to run a complete WordPress instance.

3.1 Setting up Docker

The Docker server/daemon, part of the Docker Engine, can only run under an environment that can have access to Linux kernel specific features. It is possible to run Docker Engine in other OS as long as they are capable of virtualizing Linux.

Docker has a set of tools called Docker Toolbox which allows users to run Docker in non-Linux environments. This solution installs Oracle VirtualBox, a VM hypervisor, that spins up a VirtualBox VM called *boot2docker*, containing a running Linux OS with a Docker daemon that is later used with the client in the host machine. This is a functional solution, but requires deeper knowledge about how the connection with the daemon is being made, more dependencies (such as an hypervisor) and potential issues (*boot2docker* VM needed to be updated manually for each new daemon version, VM not operating correctly after unsuspending, worse performance, etc.).

On March 2016 a new way to set up Docker in non-Linux environments was announced (Docker Blog, 2016). Assuming that the user is running a modern OS, Docker no longer needs to install an external hypervisor but uses the native ones provided by the OS (*xhyve* on macOS and *Hyper-V* on Windows machines) to yield a better experience. This option is the one being documented in this Thesis since it is now stable and the default form of installation.

4 Windows 10

The following steps show the process to install Docker Engine on a Windows 10 machine:

1. The user downloads the MSI (Microsoft Installer) file from <http://www.docker.com/products/overview>. The download link can be found in the “Install the platform” section as shown in Figure 5.

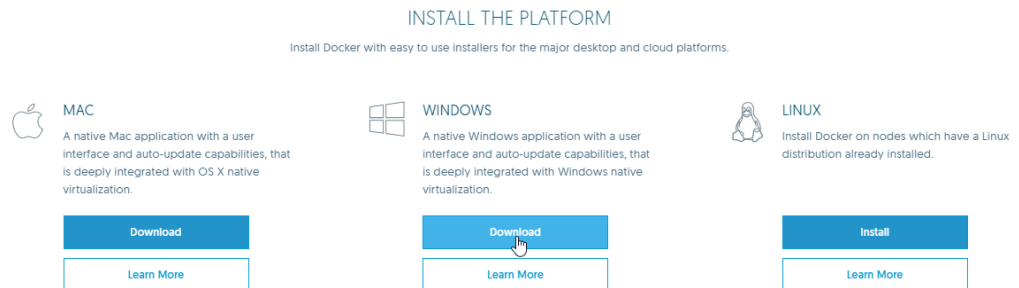


Figure 5. Docker download page (platform selection, Windows selected).

2. The user executes the MSI file and follows the steps while making sure the latest checkbox is ticked so Docker is launched after the setup is finished. Figure 6 shows an intermediate step of the setup.

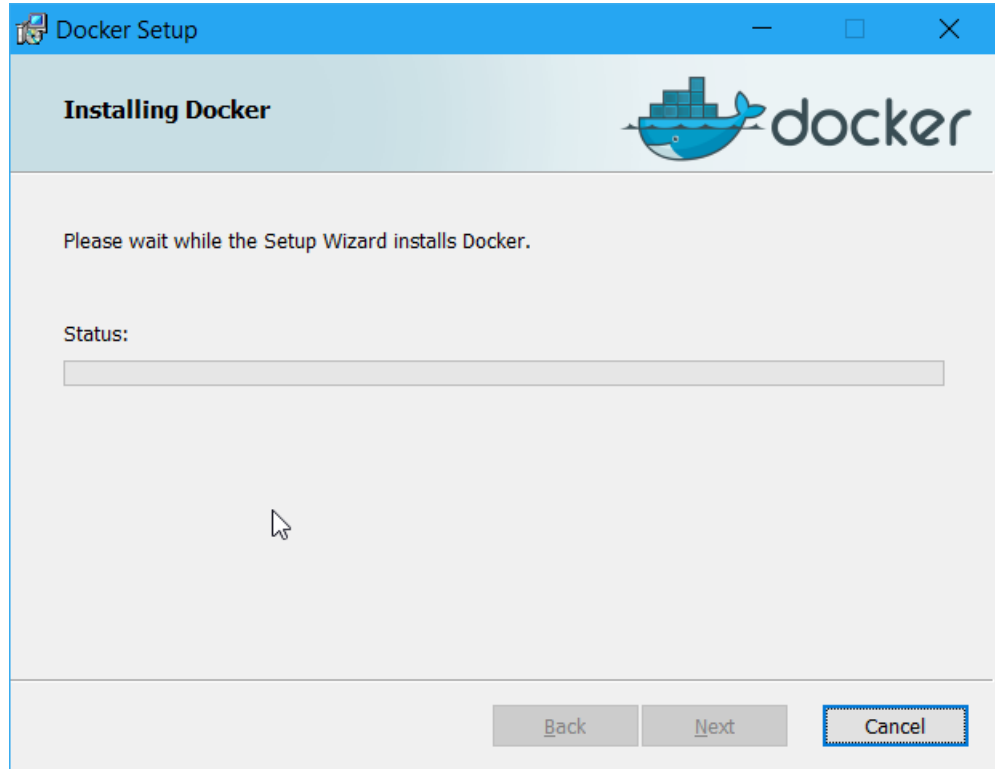


Figure 6. Docker installation assistant (Windows).

3. Once Docker is launched for the first time, it detects if Hyper-V, the native hypervisor, is enabled. If it is not, it requests the user to enable it as shown in Figure 7. The computer reboots afterwards.

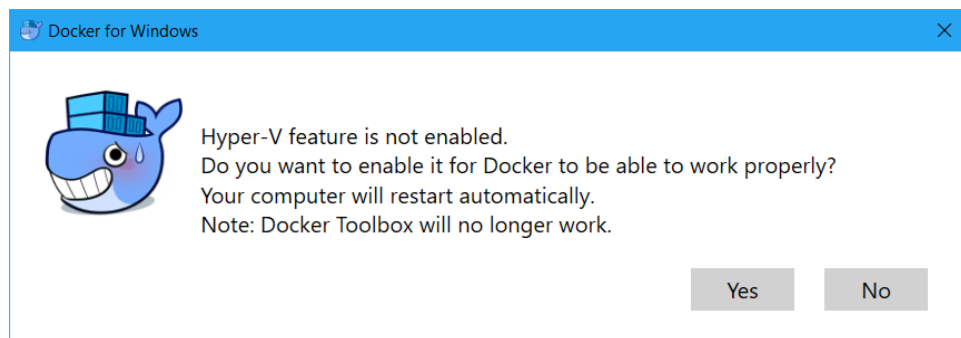


Figure 7. Docker asking to enable Hyper-V feature.

4. After reboot, Windows automatically launches Docker. The first time that this happens, it prompts the user to enable the Docker service as shown in Figure 8. After a brief period, a notification alerts the user that Docker is effectively running.

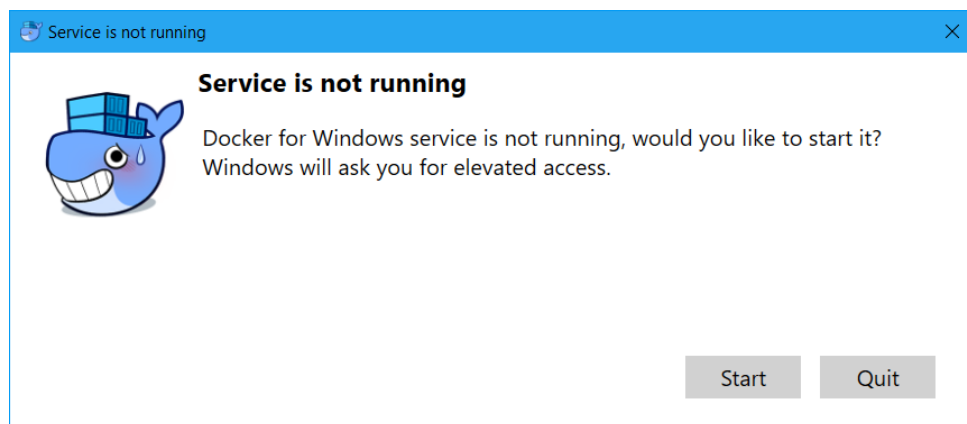
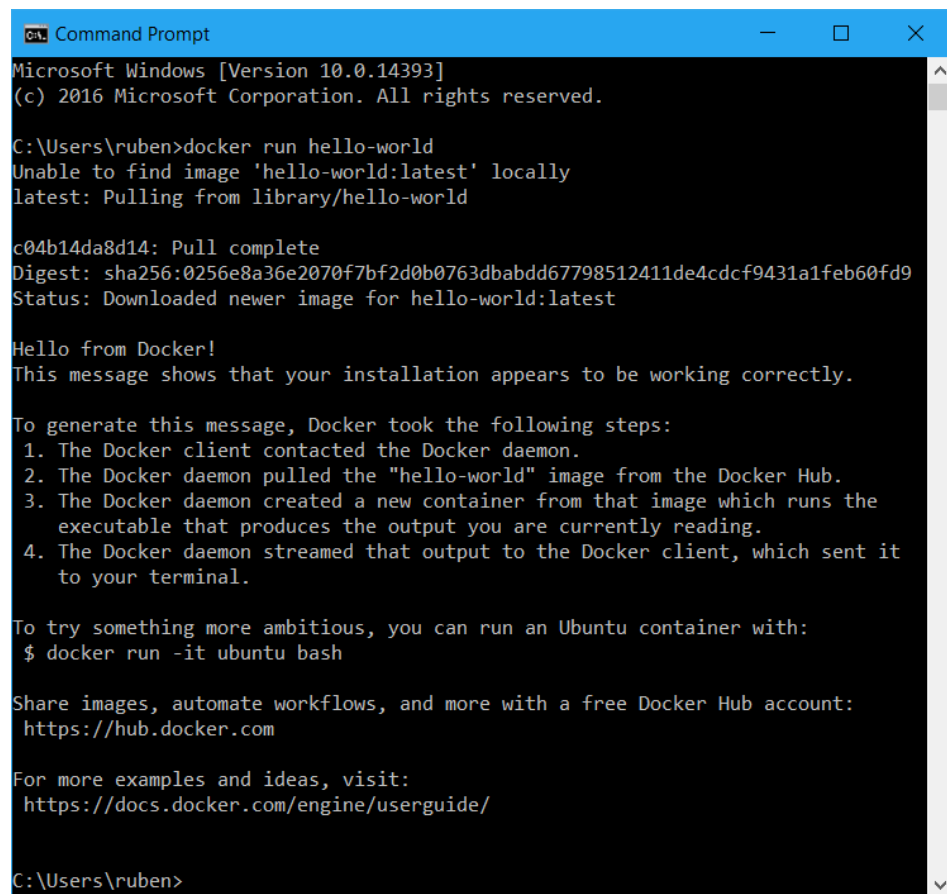


Figure 8. Docker asking to run Docker service.

5. The user can verify that Docker is working by running its first container through the Windows Command Prompt command `"docker run hello-world"`. The command calls the Docker command line interface tool to reach the Docker daemon and tries to find the container "hello-world" locally, which can not be found, so it pulls it from the registry (assuming there is internet connectivity). After all the layers have been pulled successfully, the container runs and show

an output similar to the one shown in Figure 9.



```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\ruben>docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world

c04b14da8d14: Pull complete
Digest: sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cddf9431a1feb60fd9
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker Hub account:
https://hub.docker.com

For more examples and ideas, visit:
https://docs.docker.com/engine/userguide/

C:\Users\ruben>
```

Figure 9. Running *hello-world* container (Windows).

5 macOS

The following steps show the process to install Docker Engine on a macOS machine machine:

The user downloads the dmg (Apple Disk Image) file from <http://www.docker.com/products/overview>. The download link can be found in

the “Install the platform” section as shown in Figure 10.



Figure 10. Docker download page (platform selection, Mac selected).

The user opens the dmg file. A window automatically opens as shown in Figure 11, and the user needs to drag the Docker whale icon to the folder named “Applications” situated in the same window.

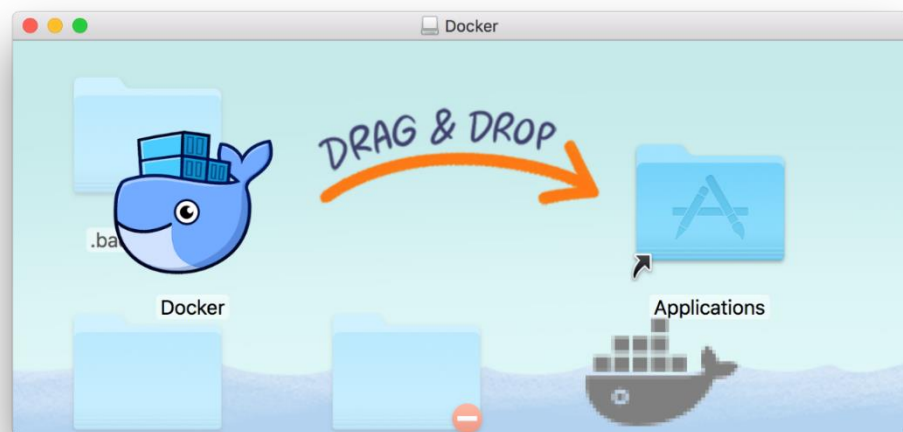


Figure 11. Installing Docker in macOS.

The user opens the installed application and receives a security alert similar to the one shown in Figure 12, as with any other newly installed application. The

user must click on “Open”.

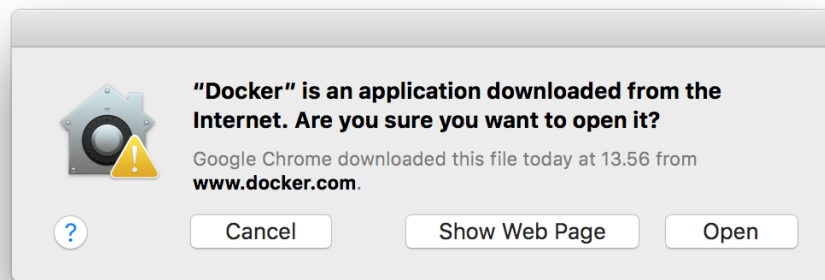


Figure 12. macOS security warning about newly downloaded Docker.

Docker welcomes the user as shown in Figure 13.

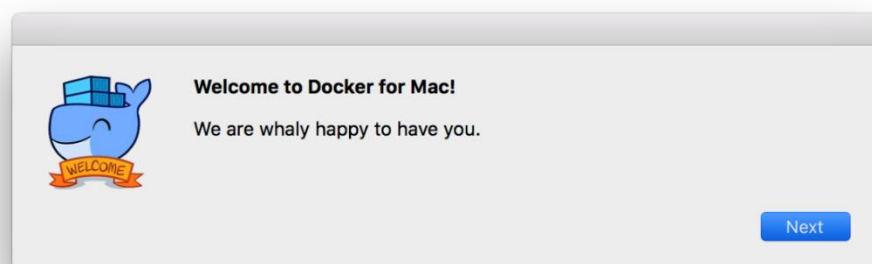


Figure 13. Docker welcome popup in macOS.

Docker requests the user privileged access to finish the setup as shown in Figure 14. The user must grant this permission by clicking “OK”.

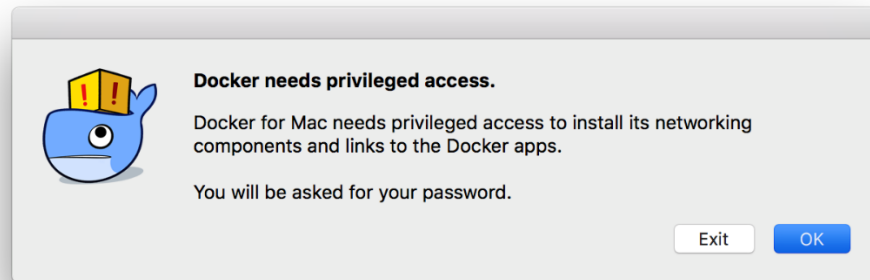
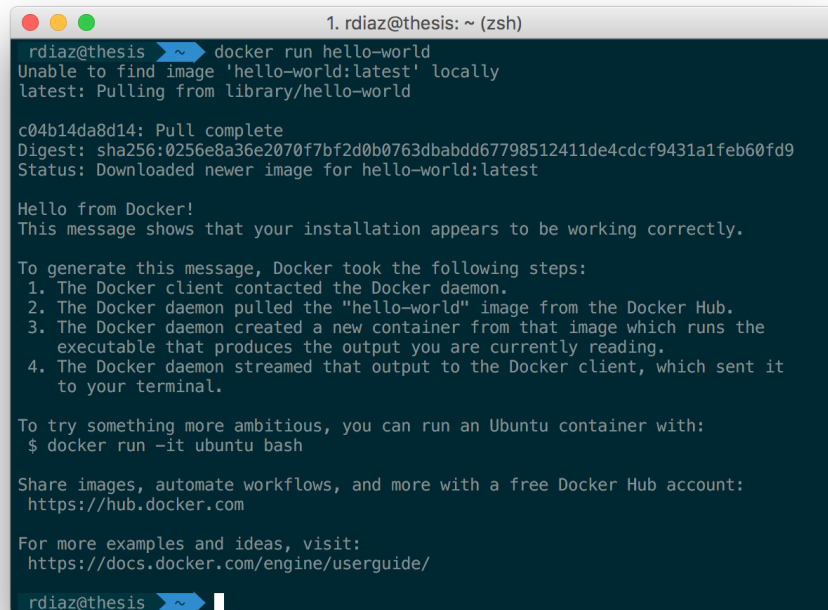


Figure 14. Docker asking for privileged access (macOS).

The user can verify that Docker is working by running its first container through any available Terminal application through the command `docker run hello-world`. An example output is shown in Figure 15 using a terminal application called *iTerm2*. The command calls the Docker command line interface tool to reach the Docker daemon and tries to find the container “hello-world” locally, which can not be found, so it pulls it from the registry (assuming there is internet connectivity). After all the layers have been pulled successfully, the container

runs.



```

rdiaz@thesis ~ (zsh)
rdiaz@thesis ~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world

c04b14da8d14: Pull complete
Digest: sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cdc9431a1feb60fd9
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker Hub account:
https://hub.docker.com

For more examples and ideas, visit:
https://docs.docker.com/engine/userguide/

rdiaz@thesis ~$

```

Figure 15. Running hello-world container (macOS).

6 Linux (Ubuntu 16.04)

The following steps show the process to install Docker Engine on a Linux machine. Ubuntu 16.04 was chosen since it is the mainstream distribution and the specific version is the LTS (Long Term Support) version, which offers support for five years and is hence the most desirable for stability.

The following instructions assume that the user has a Linux user with *sudo* privileges – which means that, the user can execute commands as superuser, also known as *root*. It also assumes working internet connectivity.

The dollar symbol (\$) in front of a line means that what it is being shown is a terminal command and such symbol must be omitted by the user typing the instructions.

1. The user must ensure the system is up-to-date. Command 1 (requires user's interactive confirmation) may be executed to update all the packages.

```
$ sudo apt-get update && sudo apt-get upgrade (1)
```

2. The user must ensure that *apt-transport-https* and *ca-certificates* packages are installed in the system. Command 2 (requires user's interactive confirmation) installs the required packages.

```
$ sudo apt-get install apt-transport-https ca-certificates (2)
```

3. The user must add the GPG (GNU Privacy Guard) key from the APT (Advanced Packaging Tool) repository in order to verify that the packages have not been manipulated – that is, they are signed by the maintainers of Docker. Command 3 (requires user's interactive confirmation) adds the new GPG key.

```
$ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADBF76221572C52609D (3)
```

4. The user must add the Docker APT repository to the system's repository list. Command 4 creates a new file under */etc/apt/sources.list.d* called *docker.list* with a line indicating the repository location

```
$ echo "deb https://apt.dockerproject.org/repo ubuntu-xenial main" | sudo tee /etc/apt/sources.list.d/docker.list (4)
```

5. The user must update the local APT package index using command 5.

```
$ sudo apt-get update (5)
```

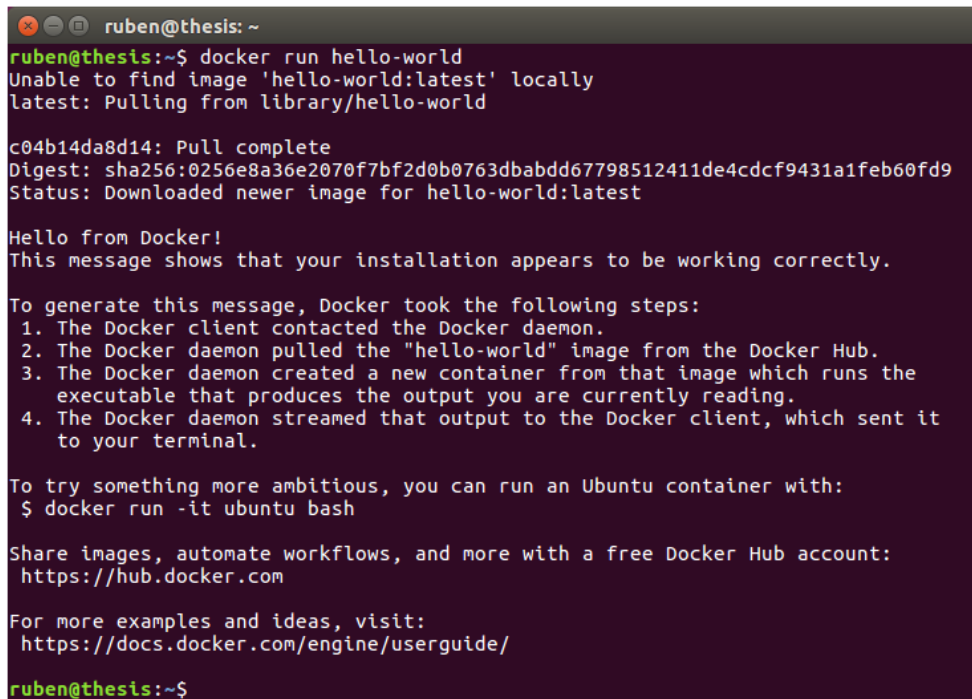
6. The user may install the recommended packages by Docker so the default storage driver aufs (Advanced Multi-Layered Unification Filesystem) can be used. Command 6 (requires user's interactive confirmation) installs the recommended packages.

```
$ sudo apt-get install linux-image-extra-$(uname -r) linux-image-extra-virtual (6)
```

The user can now install Docker Engine. Command 7 (requires user's interactive confirmation) installs Docker engine in the system:

```
$ sudo apt-get install docker-engine (7)
```

The user can verify that Docker is working by running its first container through the preferred terminal application with the command `docker run hello-world`. The command uses the Docker command line interface tool to reach the Docker daemon and try to find the container “hello-world” locally, which can not find it, so it pulls it from the registry. After all the layers have been pulled successfully, the container runs and show an output similar to the one shown in Figure 16.



```
ruben@thesis: ~  
ruben@thesis:~$ docker run hello-world  
Unable to find image 'hello-world:latest' locally  
latest: Pulling from library/hello-world  
  
c04b14da8d14: Pull complete  
Digest: sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cdcf9431a1feb60fd9  
Status: Downloaded newer image for hello-world:latest  
  
Hello from Docker!  
This message shows that your installation appears to be working correctly.  
  
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
3. The Docker daemon created a new container from that image which runs the  
   executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it  
   to your terminal.  
  
To try something more ambitious, you can run an Ubuntu container with:  
$ docker run -it ubuntu bash  
  
Share images, automate workflows, and more with a free Docker Hub account:  
https://hub.docker.com  
  
For more examples and ideas, visit:  
https://docs.docker.com/engine/userguide/  
ruben@thesis:~$
```

Figure 16. Running hello-world container (Ubuntu).

6.1 Creating a Docker container image

In order to run a container, a Docker image is required. A Docker image is composed of different layer. Docker images can have different versions, and they are hosted in a registry, which could be public or private. The most popular registry is *Docker Hub* (hub.docker.com) which allows to host public images for free and private ones free or a fee depending of the needs.

Most of the images created by users or organizations are based on ready made images available for the public at Docker Hub, as it would not make sense to reinvent the wheel every time a new image is built – that is, repeating the base steps that are shared among applications over and over.

For example, if a developer wants to build a Docker image for a Python application, the base *python* public container image (which is at the same time based on *debian* container image) could be used, which already contains a minimal functional Python runtime, and perhaps the developer could add some additional dependencies on top of that. Figure 17 shows a visual representation of this methodology.

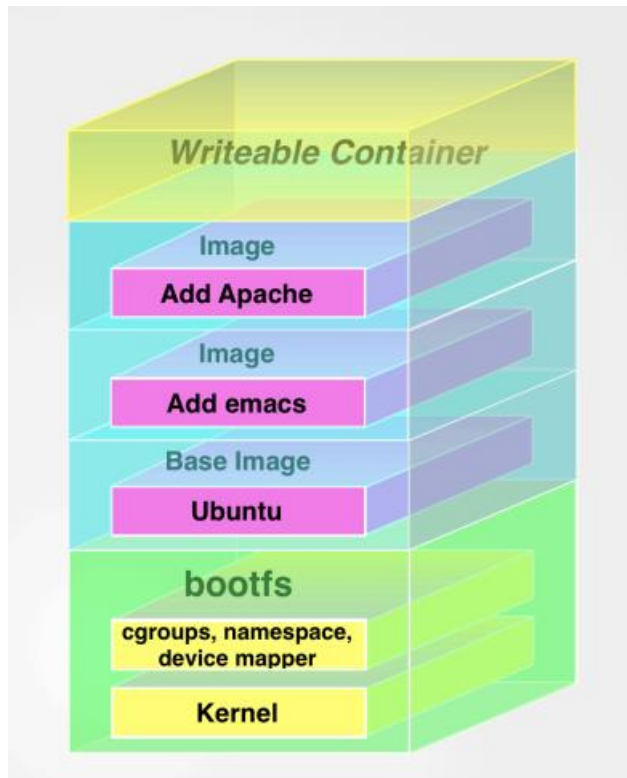


Figure 17. Container image representation (Singh A, 2015).

This model provides other users or organizations with functional base images that already contains the runtime and product they may need in some cases. Some of these base images are maintained by the organization or community that created the product or Docker Inc. itself, which implies certain guarantees of stability and continuous updates. In general, it enhances reusability since users are able to create their own images based on the already existing ones without writing every base step.

For the thesis' demonstration, the official *python* Docker base image (version 3.5.2) [https://hub.docker.com/r/_/python/] is used. This image is offered to the public for free and is maintained by Docker Inc, and is based on the official *debian* base image. The software that runs inside is a Python 3 application that exposes a public-facing HTTP (Hypertext Transfer Protocol) server, allowing the user to input some text into a basic

web form and obtain ASCII (American Standard Code for Information Interchange) art based on the input. This app has some dependencies (*aiohttp*, *cchardet* and *pyfiglet*) that are not included in the original container image and need to be installed.

The structure of the application is as follows:

```
python_app/
  Dockerfile
  requirements.txt
  server.py
  templates/
    index.html
```

The file *Dockerfile* contains instructions to generate the container image. The name itself (“Dockerfile”) is the default that the Docker command line tool uses for certain actions such as building the container image.

The file *requirements.txt* contains the list of dependencies that the Python application needs in order to run. This file is used when the container is built so that the final container image is able to run the application.

The file *server.py* contains the Python code that runs the HTTP server which handles all requests, generating the resulting ASCII art on demand.

The folder *templates* contains only a single file *index.html*, which is the HTML returned to the user when visiting the root path.

The individual content of these files is found in Appendix 1.

Given the described layout, the user is able to build the container image that contains both the Python runtime and the application itself. Running command 8 builds the container image.

```
$ docker build -t python_app . (8)
```

The command tags the container image as *python_app* for easier reference. The final dot (i.e. “current directory”) gives what it is called *context* in Docker, which means what files Docker is able to see when building the container image. All the files need to be visible for this building process, so a dot is provided.

The output of the process is similar to the one shown in Appendix 2.

The last line gives the image id that it has been just built. However, since a tag has been given to the container image (“python_app”), the user may use said tag to refer to it. To verify that the image exists, command 9 is run.

```
$ docker images (9)
```

Command 9 yields an output similar to the one shown in Figure 18.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
python_app	latest	e141ee376664	2 minutes ago	698.4 MB
python	3.5.2	302ab18dbdd0	5 days ago	682.5 MB

Figure 18. Output of command 9.

The user is now able to create a container that uses the recently created image by using command 10. The command will bind the port 8080 in the Docker daemon host to the port 8080 in the container (i.e. the app). It will also show the output of the running application, and user can stop the container anytime through the signal *SIGINT* (typically using the keyboard combination *Control + C*). For a list of arguments and flags for *docker run*, see <https://docs.docker.com/engine/reference/run/>.

```
$ docker run -p 8080:8080 -ti python_app (10)
```

The immediate output of command 10 is shown in Figure 19.

```
===== Running on http://0.0.0.0:8080 =====
(Press CTRL+C to quit)
```

Figure 19. Output of command 10.

To verify that it is running, the user may execute command 11.

```
$ docker ps (11)
```

This command lists the running containers in the Docker host machine. The output is similar to the one shown in Figure 20.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e141ee376664	python_app	"python /usr/src/app/"	4 seconds ago	Up 3 seconds	0.0.0.0:8080->8080/tcp	sad_lamarr

Figure 20. Output of command 11.

The user is able to see the application running in a browser by heading to <http://localhost:8080>. The result should be similar to what is shown in Figures 21 and 22.



Figure 21. Index page of *python_app*.

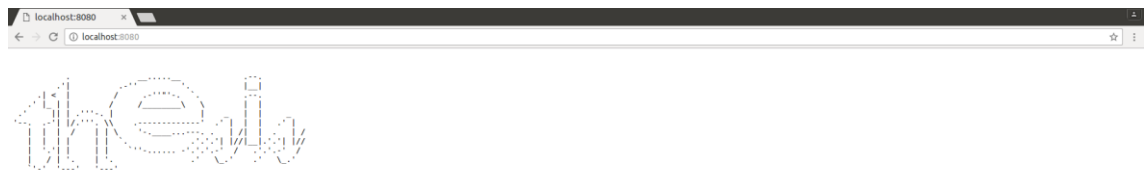


Figure 22. Example result of *python_app*.

6.2 Use case: WordPress

In this section, a complete WordPress appliance is set up using Docker and the official container images from WordPress and one of the supported databases – in this example, MySQL.

This use case shows how Docker can help to convert these tasks into easier steps due to the available public container images that can be reused for different projects.

If the user had to set up the whole WordPress typical LAMP (Linux + Apache + MySQL + PHP) stack but in a different environment than Docker, he would need to consider at least the following (Kili, 2016):

- Install and configure Apache
- Install and configure MySQL server (creating the WordPress database and user)
- Install PHP (PHP: Hypertext Preprocessor) and PHP modules (Apache, MySQL and GD support)
- Download WordPress and place it in the correct web server directory
- Set up file permissions
- Edit WordPress configuration file to match MySQL settings
- Restart Apache and MySQL to apply changes

Docker Compose

The user needs to run two containers at the same time (it is a hard dependency – WordPress will not work without a database to connect to) and link them, so one can connect to the other. Docker offers a tool called Docker Compose, which allows the user to write definitions for applications that need two or more containers in order to run. This tool is already available in the system once Docker is installed, and can be invoked through command 12.

```
$ docker-compose (12)
```

This command requires a file, which is *docker-compose.yaml* by default, to be present in the directory where this command is being executed.

Docker Compose is not strictly necessary in order to bring two containers and link them, but allows the user to organize and maintain the resources in a better way. Therefore, the best practice is used in this use case.

Writing the `docker-compose.yml`

The first step to write the Docker Compose definition is to select the container images that are going to be used and their versions. In this case, the following ones will be used:

- Container image `wordpress`, version `4.6.1-php7.0-apache`, from https://hub.docker.com/_/wordpress/
- Container image `mysql`, version `8`, from https://hub.docker.com/_/mysql/

The contents of the `docker-compose.yml` file can be found in Appendix 3.

To run the stack, command 13 is used.

```
$ docker-compose up (13)
```

The output will show errors from the WordPress container during the first initialization. This happens because the MySQL container is not yet ready as it needs to initialize the databases from scratch. Subsequent launches are much faster. The output is similar to the one shown in Appendix 4.

The user can now access the WordPress website through `http://localhost:8080` and see the WordPress installation page as shown in Figure 23.

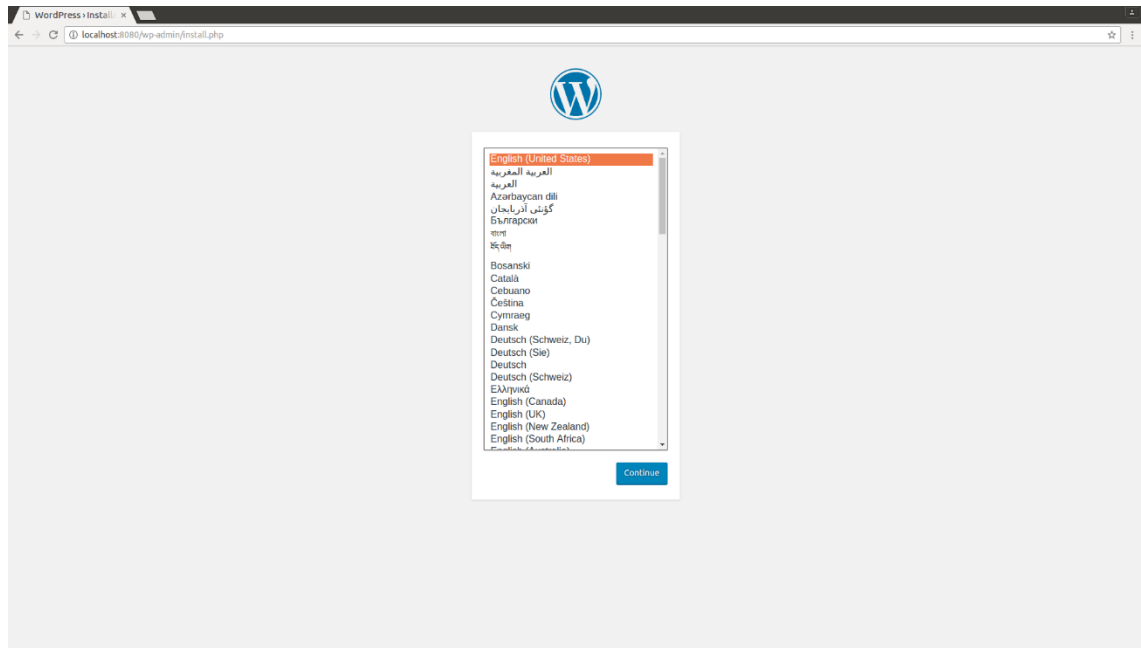


Figure 23. WordPress installation webpage.

The database settings are already supplied by the environment variables specified in the `docker-compose.yml`, hence the configuration that the user needs to supply in the installation assistant is minimal: language, site title, username, password and email. The site then will be ready to use.

This approach, which is extensible to almost any type of stack, has some advantages compared to the typical manual setup:

- User does not need to take care of the installation of each individual component and dependencies, because the container images already have everything needed to run the stack.
- The whole WordPress stack is summarized on a single file (`docker-compose.yml`). It is convenient, for example, for upgrading versions of some of the components or change the database properties.
- The `docker-compose.yml` file can be transferred to other computers with Docker Compose and it will produce the same stack with the same versions. If the local daemon already has some of the container images, it can reuse them even if they are older (only the new layers would be downloaded).

7 CONCLUSION

This Thesis has presented Docker as an emerging implementation of software containers while comparing it with Virtual Machines. It has shown Docker's main architectural parts, its image system, the setup in different OS and a walkthrough over two common practical cases.

Software containers and its implementations (such as Docker) and Virtual Machines are different since the underlying virtualization technology happens on a different level. Containers do not replace traditional hypervisor virtualization but they are a viable alternative when there is no need for full isolation or a different OS than the host OS is needed to be virtualized. Both virtualization technologies can be combined to create a hybrid approach (Sharma ym. 2016).

Docker containers take from milliseconds to few seconds to initialize, whereas Virtual Machines often take much longer. The soft limits in Docker allows users to run a large amount of containers while Virtual Machines are hard limited to the available resources. More than a convenience that could save time to the individual developer, these are critical features in CI/CD (Continuous Integration/Continuous Delivery) environments where each small change involves a workflow that implies running the stack itself each time to verify that it is stable. Since this often happens several times a day and sometimes concurrently (changes from different persons being submitted for the CI/CD workflow), this helps teams to obtain faster software iterations without needing to spend on better hardware. (Docker Inc, "CI/CD", 2017).

The Docker ecosystem is large and there is a container image for any software with minimal popularity. In the Python practical example, the official Python base image is used and only a few steps are required to be added to the resulting Dockerfile. Most of the time, the user will build his images based on already created ones, reducing the needed work to create the final image.

In the WordPress stack practical example with Docker Compose, it has been shown how Docker is useful when it comes to packaging software stacks. A single file in plain text represents a whole stack and its connections.

With these practical examples, this Thesis shows how many of the tasks that were previously reserved to system administrators (such as setting up a LAMP stack) can

now be achieved by a solo developer with no previous system administration experience, who can now focus on the application development itself.

REFERENCES

Automattic Inc. (2016). History of WordPress. Available at: <https://codex.wordpress.org/History> [Accessed 11.2.2017]

Boettiger, C. (2014). An introduction to Docker for reproducible research, with examples from the R environment. Available at: <https://arxiv.org/pdf/1410.0846.pdf> [Accessed 11.2.2017]

Conroy S. (2011). History of Virtualization. Available at: <http://www.everythingvm.com/content/history-virtualization> [Accessed 15.11.2016].

Docker Inc. (2016). Docker documentation. Available at: <https://docs.docker.com/> [Accessed 5.11.2016].

Docker Inc. (2016). Docker For Mac And Windows Beta: The Simplest Way To Use Docker On Your Laptop. Available at: <https://blog.docker.com/2016/03/docker-for-mac-windows-beta/> [Accessed 6.11.2016].

Docker Inc. (2016). Docker Hub. Available at: <https://hub.docker.com/> [Accessed 6.11.2016].

Docker Inc. (2016). Introduction to Container Security: Understanding the isolation properties of Docker. Available at: https://www.docker.com/sites/default/files/WP_IntrotoContainerSecurity_08.19.2016.pdf [Accessed 18.2.2017]

Docker Inc. (2017). CI/CD. Available at: <https://www.docker.com/use-cases/cicd> [Accessed 19.2.2017]

Docker Inc. (2017). Docker customers. Available at: <https://www.docker.com/customers#/docker-customers> [Accessed 19.2.2017]

Docker Inc. (2017). Docker Engine. Available at: <https://www.docker.com/products/docker-engine> [Accessed 18.2.2017]

Docker Inc. (2017). Docker Image Specification v1.2.0. Available at: <https://raw.githubusercontent.com/docker/docker/master/image/spec/v1.2.md> [Accessed 11.2.2017]

Docker Inc. (2017). Docker Legal Terms – Marks and logos. Available at: <https://www.docker.com/brand-guidelines> [Accessed 18.2.2017]

Docker Inc. (2017). Understand images, containers, and storage drivers. Available at: <https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/> [Accessed 18.2.2017]

Docker Inc. (2017). Understanding Docker. Available at: <https://docs.docker.com/engine/understanding-docker/> [Accessed 18.2.2017]

Docker Inc. (2017). What is Docker. Available at: <https://www.docker.com/what-docker/> [Accessed 18.2.2017].

Felter, Ferreira, Rajamony & Rubio. (2014). An Updated Performance Comparison of Virtual Machines and Linux Containers. Available at: <http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/> [Accessed 11.2.2017]

Kiili, A (2016). How to Install WordPress 4.6 On Ubuntu 16.04 Using LAMP Stack. Available at: <http://www.tecmint.com/install-wordpress-on-ubuntu-16-04-with-lamp/> [Accessed 19.2.2017]

Kleyman, B. (2012). Hypervisor 101: Understanding the Virtualization Market. Available at: <http://www.datacenterknowledge.com/archives/2012/08/01/hypervisor-101-a-look-hypervisor-market/> [Accessed 15.11.2016].

Miell & Hobson (2016). Excerpt from book "Docker in Practice", section title: "Open your Docker daemon to the world". Available at: <http://freecontent.manning.com/wp-content/uploads/docker-in-practice-the-docker-daemon.pdf> [Accessed 18.2.2017]

Oracle Corporation. (2017). MySQL 5.7 Reference Manual. Available at: <https://dev.mysql.com/doc/refman/5.7/en/introduction.html> [Accessed 11.2.2017]

Seo, Hwang, Moon, Kwon & Kim (2014). Performance Comparison Analysis of Linux Container and Virtual Machine for Building Cloud. Available at: http://onlinepresent.org/proceedings/vol66_2014/25.pdf [Accessed 18.2.2017]

Sharma, Chaufournier, Shenoy & Tay (2016). Containers and Virtual Machines at Scale: A Comparative Study. Available at: https://people.cs.umass.edu/~prateeks/papers/mw_submitted.pdf [Accessed 18.2.2017]

Singh A. (2015). Understanding Docker Container & Image. Available at: <http://collabnix.com/archives/516> [Accessed 18.2.2017]

Williams & Jackson. (2016). A Docker Fork: Talk of a Split Is Now on the Table. Available at: <http://thenewstack.io/docker-fork-talk-split-now-table/> [Accessed 11.2.2017]

Appendix 1 – Contents of the Python application files

server.py

```

from aiohttp import web
from pyfiglet import Figlet

# Handlers
async def handle(request):
    return web.Response(text=index_content, content_type="text/html")

async def handlePost(request):
    data = await request.post()
    text = data.get("text")
    font = data.get("font")
    if not text or font not in available_fonts:
        return web.Response(text="Invalid submission.", status=400)
    f = Figlet(font=font)
    return web.Response(text=f.renderText(text))

# Index content
with open("templates/index.html", "r") as f:
    index_content = f.read()

available_fonts = Figlet().getFonts()
options_string = ''.join(['<option value="{0}">{0}</option>\n'.format(f,
f) for f in available_fonts])
index_content = index_content.replace("{{OPTIONS_STRING}}",
options_string)

# Server bootstrap
app = web.Application()
app.router.add_get('/', handle)
app.router.add_post('/', handlePost)

web.run_app(app)

```

requirements.txt

```

aiohttp==1.1.1
cchardet==1.1.1
pyfiglet==0.7.5

```

Dockerfile

```

FROM python:3.5.2
EXPOSE 8080
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r requirements.txt
COPY . /usr/src/app

```

```
CMD ["python", "./server.py"]
```

index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8 />
<title>Simple ASCII art generator</title>
</head>
<body>
<h1>String to ASCII art</h1>
<form method="POST">
<input type="text" name="text" placeholder="Text" required>
<select name="font">{{OPTIONS_STRING}}</select>
<input type="submit" value="Generate art &rsaquo;">
</form>
</body>
</html>
```

Appendix 2 – Docker build output for Python application

```
Sending build context to Docker daemon 79.87 kB
Step 1 : FROM python:3.5.2
3.5.2: Pulling from library/python

43c265008fae: Pull complete
af36d2c7a148: Pull complete
143e9d501644: Pull complete
df720fc8e4f1: Pull complete
1bd9291f3483: Pull complete
d96297b6dc06: Pull complete
0a587a7fabda: Pull complete
Digest:
sha256:f729a31a199d6752b8c40d8ad5974b191a6675dd6792f997665507bc10890a7
b
Status: Downloaded newer image for python:3.5.2
----> 302ab18dbdd0
Step 2 : EXPOSE 8080
----> Running in f59f183b80fc
----> 4d439926e31e
Removing intermediate container f59f183b80fc
Step 3 : RUN mkdir -p /usr/src/app
----> Running in 3d4a31d4e114
----> ded571004c30
Removing intermediate container 3d4a31d4e114
Step 4 : WORKDIR /usr/src/app
----> Running in 415182b7a471
----> 6f579faedf1d
Removing intermediate container 415182b7a471
Step 5 : COPY requirements.txt /usr/src/app/
----> 31987ccf80e9
Removing intermediate container 96e0b5069e48
Step 6 : RUN pip install --no-cache-dir -r requirements.txt
----> Running in dab6cee1895d
Collecting aiohttp==1.1.1 (from -r requirements.txt (line 1))
  Downloading aiohttp-1.1.1-cp35-cp35m-manylinux1_x86_64.whl (153kB)
Collecting cchardet==1.1.1 (from -r requirements.txt (line 2))
  Downloading cchardet-1.1.1-cp35-cp35m-manylinux1_x86_64.whl (184kB)
Collecting pyfiglet==0.7.5 (from -r requirements.txt (line 3))
  Downloading pyfiglet-0.7.5.tar.gz (767kB)
Collecting chardet (from aiohttp==1.1.1->-r requirements.txt (line 1))
  Downloading chardet-2.3.0.tar.gz (164kB)
Collecting multidict>=2.0 (from aiohttp==1.1.1->-r requirements.txt
(line 1))
  Downloading multidict-2.1.2-cp35-cp35m-manylinux1_x86_64.whl (340kB)
Collecting async-timeout>=1.1.0 (from aiohttp==1.1.1->-r
requirements.txt (line 1))
  Downloading async_timeout-1.1.0-py3-none-any.whl
Collecting yarl>=0.5.0 (from aiohttp==1.1.1->-r requirements.txt (line
1))
  Downloading yarl-0.5.3-cp35-cp35m-manylinux1_x86_64.whl (123kB)
Installing collected packages: chardet, multidict, async-timeout,
yarl, aiohttp, cchardet, pyfiglet
  Running setup.py install for chardet: started
```

```
Running setup.py install for chardet: finished with status 'done'  
Running setup.py install for pyfiglet: started  
Running setup.py install for pyfiglet: finished with status 'done'  
Successfully installed aiohttp-1.1.1 async-timeout-1.1.0 cchardet-  
1.1.1 chardet-2.3.0 multidict-2.1.2 pyfiglet-0.7.5 yarl-0.5.3  
You are using pip version 8.1.2, however version 9.0.0 is available.  
You should consider upgrading via the 'pip install --upgrade pip'  
command.  
---> ea4c0389738b  
Removing intermediate container dab6cee1895d  
Step 7 : COPY . /usr/src/app  
---> 1b06c17b1ea8  
Removing intermediate container cef93cc0aee5  
Step 8 : CMD python /usr/src/app/server.py  
---> Running in 438e24a4f851  
---> e141ee376664  
Removing intermediate container 438e24a4f851  
Successfully built e141ee376664
```

Appendix 3 – Contents of the docker-compose.yml

```
version: '2'
services:
  mysql:
    image: mysql:8
    volumes:
      - "./data:/var/lib/mysql"
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
      MYSQL_DATABASE: wordpress
  wordpress:
    depends_on:
      - mysql
    image: wordpress:4.6.1-php7.0-apache
    links:
      - mysql
    ports:
      - "8080:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: mysql:3306
      WORDPRESS_DB_PASSWORD: wordpress
```

Appendix 4 – Output of Docker Compose

```

Creating network "wordpress_default" with the default driver
Pulling mysql (mysql:8)...
8: Pulling from library/mysql
386a066cd84a: Pull complete
827c8d62b332: Pull complete
de135f87677c: Pull complete
05822f26ca6e: Pull complete
63ddbddf6165: Pull complete
15fe0fbc587e: Pull complete
632f7facf8c4: Pull complete
a1d52de44d77: Pull complete
816ca0193bb6: Pull complete
9e85b6079977: Pull complete
6854077987aa: Pull complete
Digest:
sha256:d2a2b5be00564bc251484b8f9d3df3435225ca9b6700f45e2436185894c8f69
f
Status: Downloaded newer image for mysql:8
Pulling wordpress (wordpress:4.6.1-php7.0-apache)...
4.6.1-php7.0-apache: Pulling from library/wordpress
386a066cd84a: Already exists
269e95c6053a: Pull complete
6243d5c57a34: Pull complete
872f6d38a33b: Pull complete
e5ea5361568c: Pull complete
f81f18e77719: Pull complete
f9dbc878ca0c: Pull complete
195935e4100b: Pull complete
551fdfcfa56c: Pull complete
5fc0f8797776: Pull complete
61f4fdd1736d: Pull complete
4449a0840320: Pull complete
cb6a453ab88b: Pull complete
feb5901d37d9: Pull complete
9ea4e515c7ba: Pull complete
28264ee76117: Pull complete
afb7ef36651f: Pull complete
eeeb1d42688d: Pull complete
cc80eb0dbc6e: Pull complete
Digest:
sha256:29f9c7fb149e80cc0d38094a927ef4ddb58d3d277fe446c0c148bb0b7848b2b
7
Status: Downloaded newer image for wordpress:4.6.1-php7.0-apache
Creating wordpress_mysql_1
Creating wordpress_wordpress_1
Attaching to wordpress_mysql_1, wordpress_wordpress_1
mysql_1      | Initializing database
mysql_1      | 2016-11-12T19:08:21.879103Z 0 [Warning] TIMESTAMP with
implicit DEFAULT value is deprecated. Please use --
explicit_defaults_for_timestamp server option (see documentation for
more details).
mysql_1      | mbind: Operation not permitted
wordpress_1  | WordPress not found in /var/www/html - copying now...
mysql_1      | mbind: Operation not permitted

```

```
mysql_1      | mbind: Operation not permitted
mysql_1      | mbind: Operation not permitted
mysql_1      | 2016-11-12T19:08:24.389651Z 1 [Warning] InnoDB: New log
files created, LSN=49311
mysql_1      | 2016-11-12T19:08:25.107022Z 1 [Warning] InnoDB:
Creating foreign key constraint system tables.
wordpress_1 | Complete! WordPress has been successfully copied to
/var/www/html
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
mysql_1      | 2016-11-12T19:08:35.401772Z 0 [Warning] No existing
UUID has been found, so we assume that this is the first time that
this server has been started. Generating a new UUID: 6913c83f-a90b-
11e6-acae-0242ac130002.
mysql_1      | 2016-11-12T19:08:35.446537Z 0 [Warning] Gtid table is
not ready to be used. Table 'mysql.gtid_executed' cannot be opened.
mysql_1      | 2016-11-12T19:08:35.447450Z 4 [Warning] root@localhost
is created with an empty password ! Please consider switching off the
--initialize-insecure option.
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
```



```

wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
mysql_1      | 2016-11-12T19:09:07.502283Z 4 [Warning] 'user' entry
'mysql.sys@localhost' ignored in --skip-name-resolve mode.
mysql_1      | 2016-11-12T19:09:07.502510Z 4 [Warning] 'user' entry
'root@localhost' ignored in --skip-name-resolve mode.
mysql_1      | 2016-11-12T19:09:07.502748Z 4 [Warning] 'db' entry 'sys
mysql.sys@localhost' ignored in --skip-name-resolve mode.
mysql_1      | 2016-11-12T19:09:07.502863Z 4 [Warning] 'proxies_priv'
entry '@ root@localhost' ignored in --skip-name-resolve mode.
mysql_1      | 2016-11-12T19:09:07.503070Z 4 [Warning] 'tables_priv'
entry 'sys_config mysql.sys@localhost' ignored in --skip-name-resolve
mode.
mysql_1      | Database initialized
mysql_1      | MySQL init process in progress...
mysql_1      | 2016-11-12T19:09:21.469303Z 0 [Warning] TIMESTAMP with
implicit DEFAULT value is deprecated. Please use --
explicit_defaults_for_timestamp server option (see documentation for
more details).
mysql_1      | 2016-11-12T19:09:21.470285Z 0 [Note] mysqld (mysqld
8.0.0-dmr) starting as process 55 ...
mysql_1      | 2016-11-12T19:09:21.473585Z 0 [Note] InnoDB: Using
Linux native AIO
mysql_1      | 2016-11-12T19:09:21.473694Z 0 [Note] Plugin 'FEDERATED'
is disabled.
mysql_1      | 2016-11-12T19:09:21.474432Z 1 [Note] InnoDB: PUNCH HOLE
support available
mysql_1      | 2016-11-12T19:09:21.474448Z 1 [Note] InnoDB: Mutexes
and rw_locks use GCC atomic builtins
mysql_1      | 2016-11-12T19:09:21.474455Z 1 [Note] InnoDB: Uses event
mutexes
mysql_1      | 2016-11-12T19:09:21.474464Z 1 [Note] InnoDB: GCC
builtin __atomic_thread_fence() is used for memory barrier
mysql_1      | 2016-11-12T19:09:21.474472Z 1 [Note] InnoDB: Compressed
tables use zlib 1.2.3
mysql_1      | 2016-11-12T19:09:21.474709Z 1 [Note] InnoDB: Number of
pools: 1
mysql_1      | 2016-11-12T19:09:21.474797Z 1 [Note] InnoDB: Using CPU
crc32 instructions
mysql_1      | 2016-11-12T19:09:21.476125Z 1 [Note] InnoDB:
Initializing buffer pool, total size = 128M, instances = 1, chunk size
= 128M
mysql_1      | mbind: Operation not permitted
mysql_1      | mbind: Operation not permitted
mysql_1      | mbind: Operation not permitted
mysql_1      | mbind: Operation not permitted
mysql_1      | 2016-11-12T19:09:21.482594Z 1 [Note] InnoDB: Completed
initialization of buffer pool
mysql_1      | 2016-11-12T19:09:21.484075Z 0 [Note] InnoDB: If the
mysqld execution user is authorized, page cleaner thread priority can
be changed. See the man page of setpriority().
mysql_1      | 2016-11-12T19:09:21.565200Z 1 [Note] InnoDB: Creating
shared tablespace for temporary tables

```

```

mysql_1      | 2016-11-12T19:09:21.565342Z 1 [Note] InnoDB: Setting
file './ibtmp1' size to 12 MB. Physically writing the file full;
Please wait ...
mysql_1      | 2016-11-12T19:09:21.835849Z 1 [Note] InnoDB: File
'./ibtmp1' size is now 12 MB.
mysql_1      | 2016-11-12T19:09:21.838311Z 1 [Note] InnoDB: 96 redo
rollback segment(s) found. 96 redo rollback segment(s) are active.
mysql_1      | 2016-11-12T19:09:21.838356Z 1 [Note] InnoDB: 32 non-
redo rollback segment(s) are active.
mysql_1      | 2016-11-12T19:09:21.839583Z 1 [Note] InnoDB: 8.0.0
started; log sequence number 10691254
mysql_1      | 2016-11-12T19:09:21.843557Z 1 [Note] InnoDB: Waiting
for purge to start
mysql_1      | 2016-11-12T19:09:21.894217Z 0 [Note] InnoDB: Loading
buffer pool(s) from /var/lib/mysql/ib_buffer_pool
mysql_1      | 2016-11-12T19:09:21.902970Z 0 [Note] InnoDB: Buffer
pool(s) load completed at 161112 19:09:21
mysql_1      | 2016-11-12T19:09:22.025731Z 1 [Note] Found data
dictionary with version 1
mysql_1      | 2016-11-12T19:09:22.031215Z 0 [Warning] Failed to set
up SSL because of the following SSL library error: SSL context is not
usable without certificate and private key
mysql_1      | 2016-11-12T19:09:22.069506Z 0 [Warning] 'user' entry
'mysql.sys@localhost' ignored in --skip-name-resolve mode.
mysql_1      | 2016-11-12T19:09:22.069565Z 0 [Warning] 'user' entry
'root@localhost' ignored in --skip-name-resolve mode.
mysql_1      | 2016-11-12T19:09:22.069602Z 0 [Warning] 'db' entry 'sys
mysql.sys@localhost' ignored in --skip-name-resolve mode.
mysql_1      | 2016-11-12T19:09:22.069633Z 0 [Warning] 'proxies_priv'
entry '@ root@localhost' ignored in --skip-name-resolve mode.
mysql_1      | 2016-11-12T19:09:22.075156Z 0 [Warning] 'tables_priv'
entry 'sys_config mysql.sys@localhost' ignored in --skip-name-resolve
mode.
mysql_1      | 2016-11-12T19:09:22.126809Z 0 [Note] mysqld: ready for
connections.
mysql_1      | Version: '8.0.0-dmr' socket:
'/var/run/mysqld/mysqld.sock' port: 0 MySQL Community Server (GPL)
wordpress_wordpress_1 exited with code 1
wordpress_1  | WordPress not found in /var/www/html - copying now...
wordpress_1  | Complete! WordPress has been successfully copied to
/var/www/html
wordpress_1  |
wordpress_1  | MySQL Connection Error: (2002) Connection refused
wordpress_1  |
wordpress_1  | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1  |
wordpress_1  | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1  |
wordpress_1  | MySQL Connection Error: (2002) Connection refused
wordpress_1  |
wordpress_1  | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1  |
wordpress_1  | MySQL Connection Error: (2002) Connection refused
wordpress_1  |
wordpress_1  | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19

```

```
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
```

```

wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
mysql_1      | Warning: Unable to load
'/usr/share/zoneinfo/iso3166.tab' as time zone. Skipping it.
mysql_1      | Warning: Unable to load '/usr/share/zoneinfo/leap-
seconds.list' as time zone. Skipping it.
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
wordpress_1 |
wordpress_1 | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
wordpress_1 |
wordpress_1 | MySQL Connection Error: (2002) Connection refused
mysql_1      | Warning: Unable to load '/usr/share/zoneinfo/zone.tab'
as time zone. Skipping it.
mysql_1      | 2016-11-12T19:09:32.773334Z 6 [Warning] 'db' entry 'sys
mysql.sys@localhost' ignored in --skip-name-resolve mode.
mysql_1      | 2016-11-12T19:09:32.773402Z 6 [Warning] 'proxies_priv'
entry '@ root@localhost' ignored in --skip-name-resolve mode.
mysql_1      | 2016-11-12T19:09:32.773479Z 6 [Warning] 'tables_priv'
entry 'sys_config mysql.sys@localhost' ignored in --skip-name-resolve
mode.
mysql 1      | mysql: [Warning] Using a password on the command line
interface can be insecure.

```

```

mysql_1      | mysql: [Warning] Using a password on the command line
interface can be insecure.
mysql_1      | mysql: [Warning] Using a password on the command line
interface can be insecure.
mysql_1      | mysql: [Warning] Using a password on the command line
interface can be insecure.
mysql_1      | 2016-11-12T19:09:32.926093Z 10 [Warning] 'db' entry
'sys mysql.sys@localhost' ignored in --skip-name-resolve mode.
mysql_1      | 2016-11-12T19:09:32.926148Z 10 [Warning] 'proxies_priv'
entry '@ root@localhost' ignored in --skip-name-resolve mode.
mysql_1      | 2016-11-12T19:09:32.926453Z 10 [Warning] 'tables_priv'
entry 'sys_config mysql.sys@localhost' ignored in --skip-name-resolve
mode.
mysql_1      |
mysql_1      | 2016-11-12T19:09:32.928673Z 0 [Note] Giving 0 client
threads a chance to die gracefully
mysql_1      | 2016-11-12T19:09:32.928710Z 0 [Note] Shutting down
slave threads
mysql_1      | 2016-11-12T19:09:32.928719Z 0 [Note] Forcefully
disconnecting 0 remaining clients
mysql_1      | 2016-11-12T19:09:32.928740Z 0 [Note] Event Scheduler:
Purging the queue. 0 events
mysql_1      | 2016-11-12T19:09:32.928869Z 0 [Note] InnoDB: FTS
optimize thread exiting.
mysql_1      | 2016-11-12T19:09:33.031877Z 0 [Note] Binlog end
mysql_1      | 2016-11-12T19:09:33.033186Z 0 [Note] Shutting down
plugin 'ngram'
mysql_1      | 2016-11-12T19:09:33.033208Z 0 [Note] Shutting down
plugin 'BLACKHOLE'
mysql_1      | 2016-11-12T19:09:33.033217Z 0 [Note] Shutting down
plugin 'ARCHIVE'
mysql_1      | 2016-11-12T19:09:33.033223Z 0 [Note] Shutting down
plugin 'INNODB_CACHED_INDEXES'
mysql_1      | 2016-11-12T19:09:33.033228Z 0 [Note] Shutting down
plugin 'INNODB_SYS_VIRTUAL'
mysql_1      | 2016-11-12T19:09:33.033238Z 0 [Note] Shutting down
plugin 'INNODB_SYS_DATAFILES'
mysql_1      | 2016-11-12T19:09:33.033247Z 0 [Note] Shutting down
plugin 'INNODB_SYS_TABLESPACES'
mysql_1      | 2016-11-12T19:09:33.033256Z 0 [Note] Shutting down
plugin 'INNODB_SYS_FOREIGN_COLS'
mysql_1      | 2016-11-12T19:09:33.033264Z 0 [Note] Shutting down
plugin 'INNODB_SYS_FOREIGN'
mysql_1      | 2016-11-12T19:09:33.033270Z 0 [Note] Shutting down
plugin 'INNODB_SYS_FIELDS'
mysql_1      | 2016-11-12T19:09:33.033279Z 0 [Note] Shutting down
plugin 'INNODB_SYS_COLUMNS'
mysql_1      | 2016-11-12T19:09:33.033285Z 0 [Note] Shutting down
plugin 'INNODB_SYS_INDEXES'
mysql_1      | 2016-11-12T19:09:33.033293Z 0 [Note] Shutting down
plugin 'INNODB_SYS_TABLESTATS'
mysql_1      | 2016-11-12T19:09:33.033302Z 0 [Note] Shutting down
plugin 'INNODB_SYS_TABLES'
mysql_1      | 2016-11-12T19:09:33.033308Z 0 [Note] Shutting down
plugin 'INNODB_FT_INDEX_TABLE'
mysql_1      | 2016-11-12T19:09:33.033316Z 0 [Note] Shutting down
plugin 'INNODB_FT_INDEX_CACHE'
mysql_1      | 2016-11-12T19:09:33.033321Z 0 [Note] Shutting down
plugin 'INNODB_FT_CONFIG'

```

```

mysql_1      | 2016-11-12T19:09:33.033329Z 0 [Note] Shutting down
plugin 'INNODB_FT_BEING_DELETED'
mysql_1      | 2016-11-12T19:09:33.033338Z 0 [Note] Shutting down
plugin 'INNODB_FT_DELETED'
mysql_1      | 2016-11-12T19:09:33.033344Z 0 [Note] Shutting down
plugin 'INNODB_FT_DEFAULT_STOPWORD'
mysql_1      | 2016-11-12T19:09:33.033353Z 0 [Note] Shutting down
plugin 'INNODB_METRICS'
mysql_1      | 2016-11-12T19:09:33.033358Z 0 [Note] Shutting down
plugin 'INNODB_TEMP_TABLE_INFO'
mysql_1      | 2016-11-12T19:09:33.033367Z 0 [Note] Shutting down
plugin 'INNODB_BUFFER_POOL_STATS'
mysql_1      | 2016-11-12T19:09:33.033373Z 0 [Note] Shutting down
plugin 'INNODB_BUFFER_PAGE_LRU'
mysql_1      | 2016-11-12T19:09:33.033381Z 0 [Note] Shutting down
plugin 'INNODB_BUFFER_PAGE'
mysql_1      | 2016-11-12T19:09:33.033386Z 0 [Note] Shutting down
plugin 'INNODB_CMP_PER_INDEX_RESET'
mysql_1      | 2016-11-12T19:09:33.033394Z 0 [Note] Shutting down
plugin 'INNODB_CMP_PER_INDEX'
mysql_1      | 2016-11-12T19:09:33.033404Z 0 [Note] Shutting down
plugin 'INNODB_CMPMEM_RESET'
mysql_1      | 2016-11-12T19:09:33.033412Z 0 [Note] Shutting down
plugin 'INNODB_CMPMEM'
mysql_1      | 2016-11-12T19:09:33.033420Z 0 [Note] Shutting down
plugin 'INNODB_CMP_RESET'
mysql_1      | 2016-11-12T19:09:33.033430Z 0 [Note] Shutting down
plugin 'INNODB_CMP'
mysql_1      | 2016-11-12T19:09:33.033439Z 0 [Note] Shutting down
plugin 'INNODB_LOCK_WAITS'
mysql_1      | 2016-11-12T19:09:33.033445Z 0 [Note] Shutting down
plugin 'INNODB_LOCKS'
mysql_1      | 2016-11-12T19:09:33.033454Z 0 [Note] Shutting down
plugin 'INNODB_TRX'
mysql_1      | 2016-11-12T19:09:33.033460Z 0 [Note] Shutting down
plugin 'InnoDB'
mysql_1      | 2016-11-12T19:09:33.033536Z 0 [Note] InnoDB: Starting
shutdown...
mysql_1      | 2016-11-12T19:09:33.133894Z 0 [Note] InnoDB: Dumping
buffer pool(s) to /var/lib/mysql/ib_buffer_pool
mysql_1      | 2016-11-12T19:09:33.134621Z 0 [Note] InnoDB: Buffer
pool(s) dump completed at 161112 19:09:33
wordpress_1  |
wordpress_1  | MySQL Connection Error: (2002) Connection refused
wordpress_1  |
wordpress_1  | Warning: mysqli::__construct(): (HY000/2002):
Connection refused in - on line 19
mysql_1      | 2016-11-12T19:09:35.384835Z 0 [Note] InnoDB: Shutdown
completed; log sequence number 20316784
mysql_1      | 2016-11-12T19:09:35.389091Z 0 [Note] InnoDB: Removed
temporary tablespace data file: "ibtmp1"
mysql_1      | 2016-11-12T19:09:35.389170Z 0 [Note] Shutting down
plugin 'MRG_MYISAM'
mysql_1      | 2016-11-12T19:09:35.389183Z 0 [Note] Shutting down
plugin 'MyISAM'
mysql_1      | 2016-11-12T19:09:35.389203Z 0 [Note] Shutting down
plugin 'CSV'
mysql_1      | 2016-11-12T19:09:35.389223Z 0 [Note] Shutting down
plugin 'MEMORY'

```

```

mysql_1      | 2016-11-12T19:09:35.389230Z 0 [Note] Shutting down
plugin 'PERFORMANCE_SCHEMA'
mysql_1      | 2016-11-12T19:09:35.389291Z 0 [Note] Shutting down
plugin 'sha256_password'
mysql_1      | 2016-11-12T19:09:35.389303Z 0 [Note] Shutting down
plugin 'mysql_native_password'
mysql_1      | 2016-11-12T19:09:35.389574Z 0 [Note] Shutting down
plugin 'binlog'
mysql_1      | 2016-11-12T19:09:35.390176Z 0 [Note] mysqld: Shutdown
complete
mysql_1      |
mysql_1      |
mysql_1      | MySQL init process done. Ready for start up.
mysql_1      |
mysql_1      | 2016-11-12T19:09:35.593187Z 0 [Warning] TIMESTAMP with
implicit DEFAULT value is deprecated. Please use --
explicit_defaults_for_timestamp server option (see documentation for
more details).
mysql_1      | 2016-11-12T19:09:35.594244Z 0 [Note] mysqld (mysqld
8.0.0-dmr) starting as process 1 ...
mysql_1      | 2016-11-12T19:09:35.597657Z 0 [Note] InnoDB: Using
Linux native AIO
mysql_1      | 2016-11-12T19:09:35.597759Z 0 [Note] Plugin 'FEDERATED'
is disabled.
mysql_1      | 2016-11-12T19:09:35.598618Z 1 [Note] InnoDB: PUNCH HOLE
support available
mysql_1      | 2016-11-12T19:09:35.598648Z 1 [Note] InnoDB: Mutexes
and rw_locks use GCC atomic builtins
mysql_1      | 2016-11-12T19:09:35.598662Z 1 [Note] InnoDB: Uses event
mutexes
mysql_1      | 2016-11-12T19:09:35.598679Z 1 [Note] InnoDB: GCC
builtin __atomic_thread_fence() is used for memory barrier
mysql_1      | 2016-11-12T19:09:35.598722Z 1 [Note] InnoDB: Compressed
tables use zlib 1.2.3
mysql_1      | 2016-11-12T19:09:35.599340Z 1 [Note] InnoDB: Number of
pools: 1
mysql_1      | 2016-11-12T19:09:35.599494Z 1 [Note] InnoDB: Using CPU
crc32 instructions
mysql_1      | 2016-11-12T19:09:35.601967Z 1 [Note] InnoDB:
Initializing buffer pool, total size = 128M, instances = 1, chunk size
= 128M
mysql_1      | mbind: Operation not permitted
mysql_1      | mbind: Operation not permitted
mysql_1      | mbind: Operation not permitted
mysql_1      | mbind: Operation not permitted
mysql_1      | 2016-11-12T19:09:35.616239Z 1 [Note] InnoDB: Completed
initialization of buffer pool
mysql_1      | 2016-11-12T19:09:35.618588Z 0 [Note] InnoDB: If the
mysqld execution user is authorized, page cleaner thread priority can
be changed. See the man page of setpriority().
mysql_1      | 2016-11-12T19:09:35.684594Z 1 [Note] InnoDB: Creating
shared tablespace for temporary tables
mysql_1      | 2016-11-12T19:09:35.684755Z 1 [Note] InnoDB: Setting
file './ibtmp1' size to 12 MB. Physically writing the file full;
Please wait ...
mysql_1      | 2016-11-12T19:09:35.976839Z 1 [Note] InnoDB: File
'./ibtmp1' size is now 12 MB.
mysql_1      | 2016-11-12T19:09:35.978767Z 1 [Note] InnoDB: 96 redo
rollback segment(s) found. 96 redo rollback segment(s) are active.

```

```

mysql_1      | 2016-11-12T19:09:35.978807Z 1 [Note] InnoDB: 32 non-
redo rollback segment(s) are active.
mysql_1      | 2016-11-12T19:09:35.979531Z 1 [Note] InnoDB: 8.0.0
started; log sequence number 20316784
mysql_1      | 2016-11-12T19:09:35.983259Z 1 [Note] InnoDB: Waiting
for purge to start
mysql_1      | 2016-11-12T19:09:36.034290Z 0 [Note] InnoDB: Loading
buffer pool(s) from /var/lib/mysql/ib_buffer_pool
mysql_1      | 2016-11-12T19:09:36.040981Z 0 [Note] InnoDB: Buffer
pool(s) load completed at 161112 19:09:36
mysql_1      | 2016-11-12T19:09:36.144443Z 1 [Note] Found data
dictionary with version 1
mysql_1      | 2016-11-12T19:09:36.149846Z 0 [Warning] Failed to set
up SSL because of the following SSL library error: SSL context is not
usable without certificate and private key
mysql_1      | 2016-11-12T19:09:36.150856Z 0 [Note] Server hostname
(bind-address): '*'; port: 3306
mysql_1      | 2016-11-12T19:09:36.150896Z 0 [Note] IPv6 is available.
mysql_1      | 2016-11-12T19:09:36.150921Z 0 [Note] - '::' resolves
to '::';
mysql_1      | 2016-11-12T19:09:36.150976Z 0 [Note] Server socket
created on IP: '::'.
mysql_1      | 2016-11-12T19:09:36.188266Z 0 [Warning] 'db' entry 'sys
mysql.sys@localhost' ignored in --skip-name-resolve mode.
mysql_1      | 2016-11-12T19:09:36.188315Z 0 [Warning] 'proxies_priv'
entry '@ root@localhost' ignored in --skip-name-resolve mode.
mysql_1      | 2016-11-12T19:09:36.194562Z 0 [Warning] 'tables_priv'
entry 'sys_config mysql.sys@localhost' ignored in --skip-name-resolve
mode.
mysql_1      | 2016-11-12T19:09:36.245443Z 0 [Note] mysqld: ready for
connections.
mysql_1      | Version: '8.0.0-dmr' socket:
'/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server
(GPL)
wordpress_1 | AH00558: apache2: Could not reliably determine the
server's fully qualified domain name, using 172.19.0.3. Set the
'ServerName' directive globally to suppress this message
wordpress_1 | AH00558: apache2: Could not reliably determine the
server's fully qualified domain name, using 172.19.0.3. Set the
'ServerName' directive globally to suppress this message
wordpress_1 | [Sat Nov 12 19:09:38.136695 2016] [mpm_prefork:notice]
[pid 1] AH00163: Apache/2.4.10 (Debian) PHP/7.0.12 configured --
resuming normal operations
wordpress_1 | [Sat Nov 12 19:09:38.136730 2016] [core:notice] [pid 1]
AH00094: Command line: 'apache2 -D FOREGROUND'

```