WeThinkCode_

Dev Ops

Project III

Docker:
Now, you're thinkinking with containers...

Developer
Sibonelo Nkosi
Username: sinkosi

October 2020

CONTENTS

1 SUMMARY

Right now, you are probably developing a one-block app, with softwares and libraries installed directly in your development environment, or maybe in a virtual environment... **WeThinkCode_** has provided students with two options.

IMAGINE IF YOUR APPLICATION HAS TO BE DEPLOYED ALL OVER THE WORLD AND YOU HAVE TO RE- DEVELOP IT FOR ALL EXISTING PLATFORMS AND OS...

Docker was created to satisfy this need for unification and normalisation: it makes it possible to split an application into several microservices, light, adaptable, universal and scalable, and it also gives the system administrators a great flexibility to deploy and scale up the app. This suite of projects on Docker will help you better understand this specific tool, but also the various aspects of applications development using microservices.

The aim of the Docker-1 project is to make you handle docker and docker-machine, the bases to understand the idea of containerization of services. You can see this project as an initiation.

2 GETTING STARTED

2.1 Windows

Windows Installation[*]: Windows is trash[1]

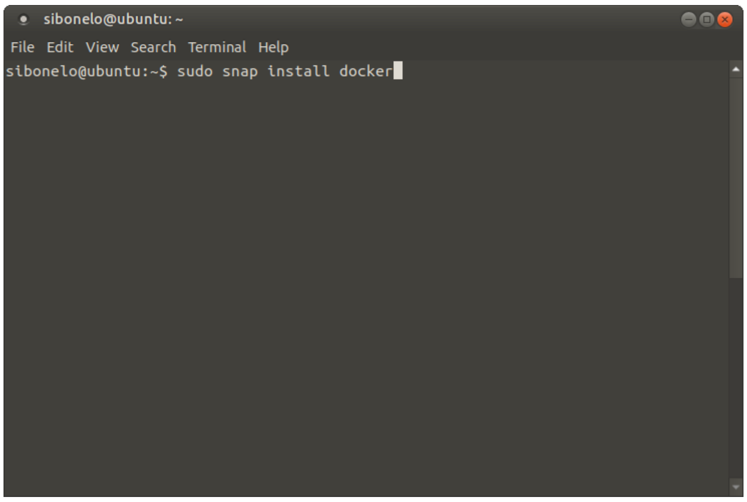
2.2 Linux

Linux Installation[**]: Begin by ensuring that you have docker[2] installed on your system, if not type:

```
$ sudo snap install docker
```

2.3 MacOS

At the time of typing this document a Mac was not available to conduct testing but the documentation[3] does state that installation and



```
sibonelo@ubuntu:~$ sudo snap install docker
```

Figure 1: *sudo snap install*, of Docker on Ubuntu

setup occurs with a call to *Homebrew* or an installation of Docker Desktop.

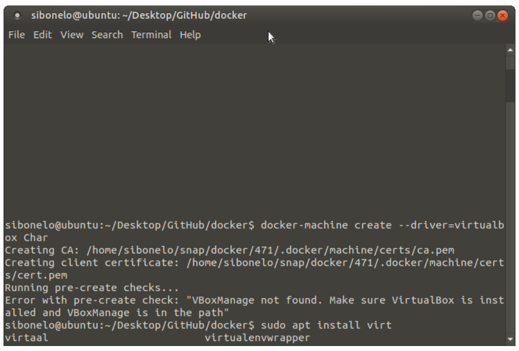
3 OO_HOW_TO_DOCKER

3.1 Exercise 01 - 09

01 - Create docker-machine

```
docker—machine create ——driver=virtualbox Char
```

This will begin the creation of the machine. Ensure that you have installed 'virtualbox' or an error will be received.


sibonelo@ubuntu: ~/Desktop/GitHub/docker
File Edit View Search Terminal Help

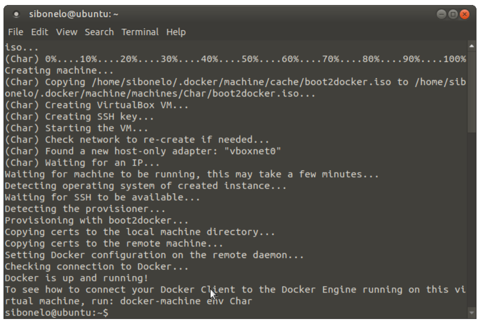
```
sibonelo@ubuntu:~/Desktop/GitHub/docker$ docker-machine create --driver=virtualb
ox Char
Creating CA: /home/sibonelo/snap/docker/471/.docker/machine/certs/ca.pem
Creating client certificate: /home/sibonelo/snap/docker/471/.docker/machine/cert
s/cert.pem
Running pre-create checks...
Error with pre-create check: "VBoxManage not found. Make sure VirtualBox is inst
alled and VBoxManage is in the path"
sibonelo@ubuntu:~/Desktop/GitHub/docker$ sudo apt install virt
virtaal                          virtualenvwrapper
```

Figure 2: Install error *sudo user*, for VirtualBox

* *Information provided is correct for current users configuration i.e Windows Home 10:2004, results may differ for other configurations*
** *Snap install is not available for all Linux Distros, this is expected to work on Ubuntu and Debian flavours*

NB: The snap install will cause virtualbox and VBoxManage to not work, so rather uninstall it and use the official installation and instructions LinuxHint[4]

If everything is configured correctly, the terminal will begin downloading an ISO. Reference to Figure 3.



```
iso...
(Char) 0%....10%....20%....30%....40%....50%....60%....70%....80%....90%....100%
Creating machine...
(Char) Copying /home/sibonelo/.docker/machine/cache/boot2docker.iso to /home/sib
onelo/.docker/machine/machines/Char/boot2docker.iso...
(Char) Creating VirtualBox VM...
(Char) Creating SSH key...
(Char) Starting the VM...
(Char) Check network to re-create if needed...
(Char) Found a new host-only adapter: "vboxnet0"
(Char) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this vi
rtual machine, run: docker-machine env Char
sibonelo@ubuntu:~$
```

Figure 3: Install *success*, for Docker-Machine - Char

You have successfully created a Docker Machine named 'Char'.

02 - Get docker-machine IP

To get the ip address of the docker-machine '**Char**' that we have just created needs us to type:

```
docker-machine ip Char
```

The address that is returned for our current machine is:

192.168.99.100

At this point I would recommend running:

```
docker-machine ls
```

This will list all the docker-machines currently installed on your system and show you their status.

03 - Set Environment Variables

In order to set the env for your new machine, you can follow the links below. The reason for doing so, is to allow docker-compose to know what to load in the yaml file before running. This will all be explained later. ENV values are available to containers[5], but also RUN-style commands during the Docker build starting with the line where they are introduced.

This is best explained in the documentation[6] which can be found at `https://docs.docker.com/machine/get-started/`. The subsection is section 6, connect your shell to the new machine

The command to set env for docker-machine 'Char'

```
eval "$(docker-machine env Char)"
```

04 - hello_world Image

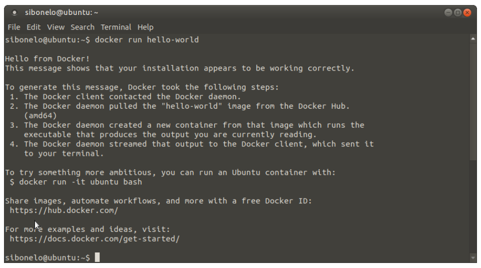
Get the 'hello_world' image from Docker Hub[7] and pull it with this command:

```
docker pull hello-world
```

05 - Run Image

Launch the hello-world container:

```
docker run hello-world
```



```
sibonelo@ubuntu:~$ docker run hello-world

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.
    (amd64)
 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 ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

sibonelo@ubuntu:~$
```

Figure 4: Running hello_world image

06 - Launch an Nginx container

To launch an NginX container, you first have to make sure you have it installed. Check if nginx is listed[8]:

```
docker image ls
```

If it is not listed, run:

```
docker pull nginx
```

After Nginx is installed as an image, run the below commands:

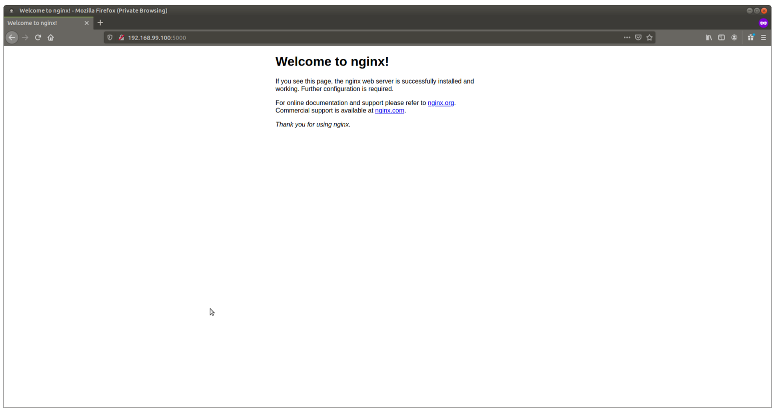
```
docker run -d -p 5000:80 --name overlord
 --restart=always nginx
```

Explainer[9]:

- -d: Run in background
- -p: Set port (5000:80 is a configuration)

- –name: Set name to overlord
- restart=always: Tells docker to always try restart the container

Remember the IP address retrieved in Step 2, if setup has gone correctly visit the ip address and port.
For me this is: `http://192.168.99.100:5000/` Your browser should look like Figure 5



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Figure 5: Running Nginx server image on `http://192.168.99.100:5000/`.

NB: Check if it is 'http' or 'https', I wasted quite a few minutes and the best of my curse words on that small discrepancy.

07 - Get the internal IP address of the overlord container

The network above allows you a **bridge** into the internal workings of Docker. The question now is, how do you address the internal Nginx server itself, more specifically... 'overlord':

```
docker network ls
```

is a good place to start. As explained on freecodecamp.org[10], you will then proceed to use the inspect command. You must know the name of the object you wish to inspect.

```
docker inspect -f "{{␣.NetworkSettings.IPAddress␣}}"
    overlord
```

-f = Format the output given the template
The extensive commandline is available here: `https://docs.docker.com/engine/reference/commandline/inspect/`

My returned ip-address is: 172.17.0.2

08 - Launch a shell from an alpine container

To launch an *Alpine* container, you first have to make sure you have it installed. Check if alpine is listed[8]:

```
docker image ls
```

If it is not listed, run:

```
docker pull alpine
```

After alpine is installed as an image, run the below commands:

```
docker run -i -t --rm alpine
```

Explainer

- d=false: Detached mode: Run container in the background, print new container id
- -i: Keep STDIN open even if not attached[11]
- -t: Allocate a pseudo-tty[11]
- –rm=false: Automatically remove the container when it exits[12]
- restart=always: Tells docker to always try restart the container

09 - From the shell of a debian container

To launch a *Debian* container, you first have to make sure you have it installed. Check if Debian is listed[8]:

```
docker image ls
```

If it is not listed, run:

```
docker pull debian
```

After debian is installed as an image, run the below commands:

```
docker run -i -t --rm debian
```

Run a nested call to update, upgrade and install gcc & git:

```
apt-get update -y && apt-get upgrade -y
&& apt-get install gcc -y &&
apt-get install git -y
```

This should provide your image the ability to compile C code and push to a git.

3.2 Exercise 10 - 19

10. Create a volume named hatchery

Creating a Volume in Docker is key to taking your applications to production[13] To do this one must run a volume create command.

```
docker volume create  hatchery
```

'hatchery' is the name given to our volume that we created.

11. List all the Docker volumes created on the machine

To check if you were successful in creating your volume or to check all the volumes installed on your machine, run[13]:

```
docker volume ls
```

This shows **ONLY** volumes.

12. Launch a mysql container as a background task

This is quite challenging and needs an individual to be focused, not because the task is large but because of the details embedded within.

'spawning-pool' -> 'hatchery' -> 'mySql' -> 'zerglings' -> 'root:Kerrigan'

To launch a *mysql* container, you first have to make sure you have it installed. Check if MySQL is listed[8]:

```
docker image ls
```

If it is not listed, run:

```
docker pull mysql
```

After mysql is installed as an image, run the below commands:

```
docker run --name=spawning-pool -d
    --env="MYSQL_ROOT_PASSWORD=Kerrigan"
    --env="MYSQL_DATABASE=zerglings"
    --volume=hatchery:/var/lib/mysql
    --restart=on-failure mysql
```

Explainer[14]:[15]

- -d: Run in background
- -e: Set Environment Variables
- -v: Set Volume to use
- –name: Set name to desired [variable]

- –restart=on-failure: Tells docker to try restart the container if it **fails**

It is worth noting that this matter is only complex because of how the statement concatenates together. If you check the references posted above, it is taken step-by-step. The order I cannot stress this enough of the arguments is very important. At a minimum it will not build what you want, worst case is it builds unusable containers. Take caution! It will restart only on failure.

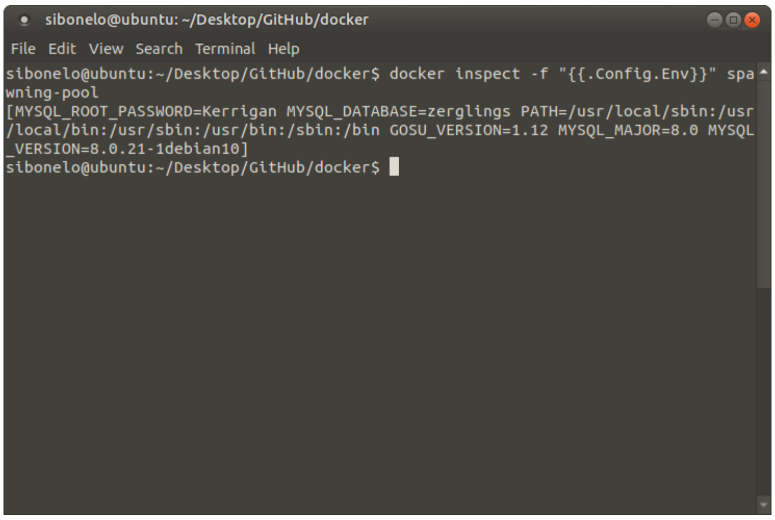
13. Print the environment variables of the spawning-pool

To see if everything worked correctly simply print its environment variables. To do this run:

```
docker inspect −f "{{.Config.Env}}" spawning−pool
```

If spawning-pool failed to set, you will receive an error container in one command, otherwise your output should be similar to Figure 6 to be sure that you have configured your container properly.

We used a similar 'inspect' method earlier. Please reference back to it if you are lost[5].



sibonelo@ubuntu: ~/Desktop/GitHub/docker
File Edit View Search Terminal Help

```
sibonelo@ubuntu:~/Desktop/GitHub/docker$ docker inspect -f "{{.Config.Env}}" spa
wning-pool
[MYSQL_ROOT_PASSWORD=Kerrigan MYSQL_DATABASE=zerglings PATH=/usr/local/sbin:/usr
/local/bin:/usr/sbin:/usr/bin:/sbin:/bin GOSU_VERSION=1.12 MYSQL_MAJOR=8.0 MYSQL
_VERSION=8.0.21-1debian10]
sibonelo@ubuntu:~/Desktop/GitHub/docker$
```

Figure 6: Print out of MySQL container env variables

14. Launch a wordpress container as a background task

To launch a *wordpress* container, you first have to make sure you have it installed. Check if WordPress is listed[8]:

```
docker image ls
```

If it is not listed, run:

```
    docker pull wordpress
```

After WordPress is installed as an image, run the below commands:

```
docker run -d -p 8080:80 --name lair
    --link spawning-pool wordpress
```

We used a similar 'inspect' method earlier. Please reference back to it if you are lost[5].

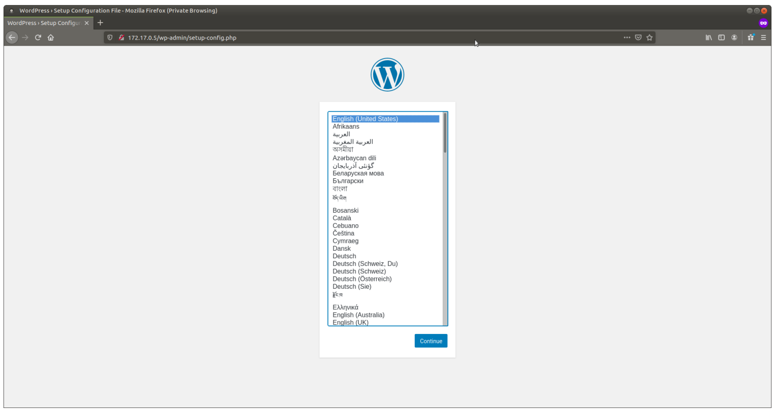


Figure 7: Running WordPress container linked to MySQL 'spawning-pool'

Explainer[16]

- -d: Run in background
- -p: Set port (8080:80 is a configuration)
- –link: Sets the two or more containers to be linked
- –name: Set name to desired [variable]

The name is then set to 'lair'. Run the command to get the IP of the container

```
docker inspect -f "{{␣.NetworkSettings.IPAddress␣}}" lair
```

and proceed to that address with the 8080 port appended.

15. Launch a phpmyadmin container as a background task

To launch a *phpmyadmin* container, you first have to make sure you have it installed. Check if phpmyadmin is listed[8]:

```
    docker image ls
```

If it is not listed, run:

```
    docker pull phpmyadmin
```

After phpmyadmin is installed as an image, run the below commands[17]:

```
docker run  -d --name roach-warden -p 8081:80
    --link spawning-pool:db phpmyadmin
```

Check for the ip address and visit the link with the port activated. You should see a screen similar to Figure 8

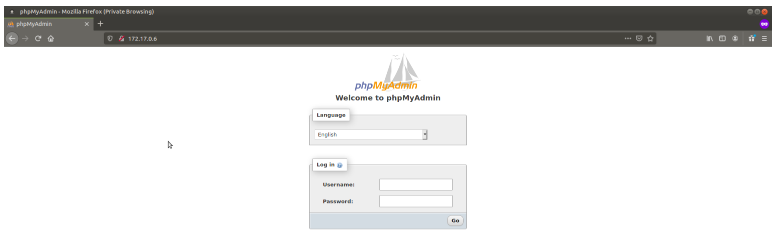


Figure 8: Running PHPMyAdmin container linked to MySQL DB 'spawning-pool'

16. Look up the spawning-pool container's logs

To see the log run:

```
docker logs -f spawning-pool
```

There are other variations that one can try though[18].

17. Display all the currently active containers on the Char virtual machine

To list the status of you containers run:

```
docker stats
```

expect an output similar as shown in Figure 9 on the following page

To list the active containers run:

```
docker ps
```

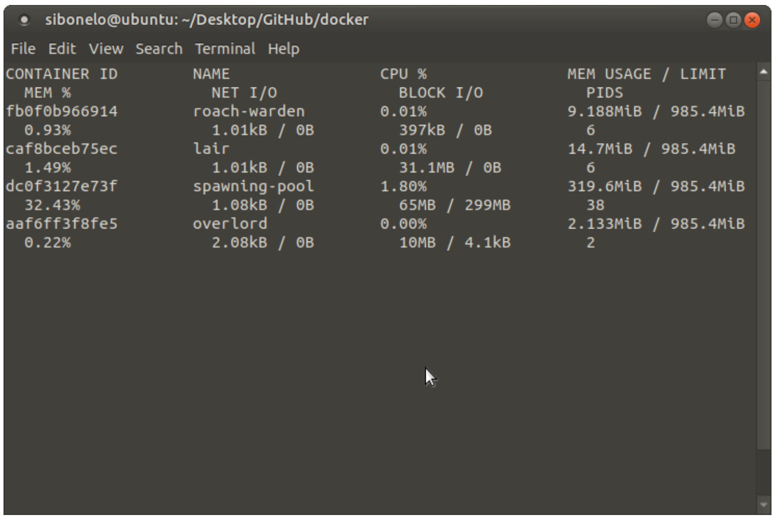
expect an output similar as shown in Figure 10 on the next page

18. Relaunch the overlord container

To restart a container needs you to tell docker to restart using the container's name e.g:

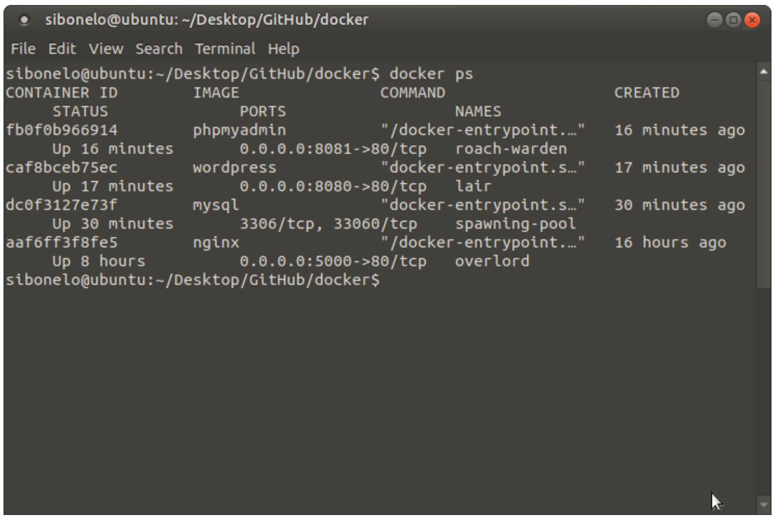
```
docker restart <container-name>
```

To restart overlord simply run:


sibonelo@ubuntu: ~/Desktop/GitHub/docker
File Edit View Search Terminal Help

```
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
  MEM %               NET I/O             BLOCK I/O           PIDS
fb0f0b966914        roach-warden        0.01%               9.188MiB / 985.4MiB
  0.93%               1.01kB / 0B         397kB / 0B          6
caf8bceb75ec        lair                0.01%               14.7MiB / 985.4MiB
  1.49%               1.01kB / 0B         31.1MB / 0B         6
dc0f3127e73f        spawning-pool       1.80%               319.6MiB / 985.4MiB
  32.43%              1.08kB / 0B         65MB / 299MB        38
aaf6ff3f8fe5        overlord            0.00%               2.133MiB / 985.4MiB
  0.22%               2.08kB / 0B         10MB / 4.1kB        2
```

Figure 9: Running 'docker stats' to ascertain running containers


sibonelo@ubuntu: ~/Desktop/GitHub/docker
File Edit View Search Terminal Help

```
sibonelo@ubuntu:~/Desktop/GitHub/docker$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
     STATUS              PORTS                  NAMES
fb0f0b966914        phpmyadmin          "/docker-entrypoint.…"   16 minutes ago
     Up 16 minutes       0.0.0.0:8081->80/tcp   roach-warden
caf8bceb75ec        wordpress           "docker-entrypoint.s…"   17 minutes ago
     Up 17 minutes       0.0.0.0:8080->80/tcp   lair
dc0f3127e73f        mysql               "docker-entrypoint.s…"   30 minutes ago
     Up 30 minutes       3306/tcp, 33060/tcp    spawning-pool
aaf6ff3f8fe5        nginx               "/docker-entrypoint.…"   16 hours ago
     Up 8 hours          0.0.0.0:5000->80/tcp   overlord
sibonelo@ubuntu:~/Desktop/GitHub/docker$
```

Figure 10: Running 'docker ps' to see container processes

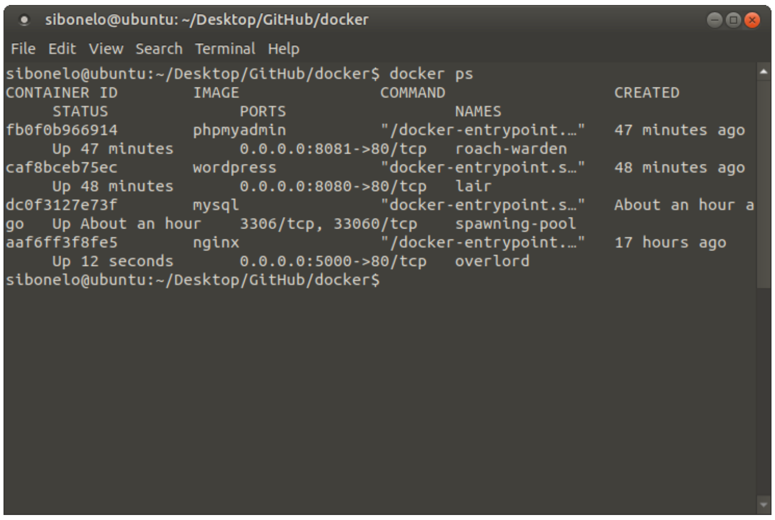
```
docker restart overlord
```

You may notice that 'overload' now has a different uptime. It has gone from '8 hours' as shown in Figure 10 to a few seconds as shown in Figure 11 on the next page.

To learn more about the docker commandline please reference as before to the Docker Documentation[8].

19. Launch a container name Abathur

. It will be a Python container, 2-slim version, its /root folder will be bound to a HOME folder on your host, and its 3000 port will be bound to the 3000 port of your virtual machine. You will personalize t


sibonelo@ubuntu: ~/Desktop/GitHub/docker
File Edit View Search Terminal Help

```
sibonelo@ubuntu:~/Desktop/GitHub/docker$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
     STATUS              PORTS                  NAMES
fb0f0b966914        phpmyadmin          "/docker-entrypoint.…"   47 minutes ago
     Up 47 minutes       0.0.0.0:8081->80/tcp   roach-warden
caf8bceb75ec        wordpress           "docker-entrypoint.s…"   48 minutes ago
     Up 48 minutes       0.0.0.0:8080->80/tcp   lair
dc0f3127e73f        mysql               "docker-entrypoint.s…"   About an hour a
go   Up About an hour    3306/tcp, 33060/tcp    spawning-pool
aaf6ff3f8fe5        nginx               "/docker-entrypoint.…"   17 hours ago
     Up 12 seconds       0.0.0.0:5000->80/tcp   overlord
sibonelo@ubuntu:~/Desktop/GitHub/docker$
```

Figure 11: Running 'docker ps' to see the restart of 'overlord'

4 01_DOCKERFILES

A Section or subsection covering extensively unit testing will be key either here or on it's own chapter

5 RESULTS AND DISCUSSION

6 CODE SNIPPET AS A FIGURE

Reference to Figure 12.

```
import { Button } from "@material-ui/core"
import React, { useState } from "react"
import GameCanvas from "./GameCanvas"

const Game = (props) => {
  const [gameOver, setGameOver] = useState(true)

  const handleStart = () => setGameOver(false)

  return (
    <>
      {gameOver ? (
        <Button color="primary" onClick={handleStart}>
          start
        </Button>
      ) : (
        <GameCanvas setGameOver={setGameOver} />
      )}
    </>
  )
}

export default Game
```

Listing 1: Game Start Code.

Figure 12: Code Snippet from the game front-end to get it running *localhost limitation*, JSX Component, from `http://localhost:3000/`.

7 BIBLIOGRAPHY

REFERENCES

[1] Docker Documentation Website. Install docker on windows 10 home. *https://docs.docker.com/docker-for-windows/faqs/#can-i-install-docker-desktop-on-windows-10-home*, Current Version, 2020.

[2] Docker Documentation Website. Install docker on linux. *https://docs.docker.com/engine/install/ubuntu/*, Current Version, 2020.

[3] Docker Documentation Website. Install docker on macos. *https://docs.docker.com/docker-for-mac/install/#install-and-run-docker-desktop-on-mac*, Current Version, 2020.

[4] Shahriar Shovon. How to setup docker machine with virtualbox. *https://linuxhint.com/setup_docker_machine_virtualbox/*, 2019.

[5] vsupalov. Docker arg, env and .env - a complete guide. *https://vsupalov.com/docker-arg-env-variable-guide/*.

[6] Docker Website. Getting started. *https://docs.docker.com/machine/get-started/*, (Section 6), 2020.

[7] Docker Hub. Docker official images: hello world. *https://hub.docker.com/_/hello-world*, 2020.

[8] Docker Docs. Docker commandline. *https://docs.docker.com/engine/reference/commandline/image_ls/*, 2020.

[9] Docker Documentation. Docker: Start containers automatically. *https://docs.docker.com/config/containers/start-containers-automatically/*, 2020.

[10] Marcelo Costa. How to get a docker container ip address - explained with examples. *https://www.freecodecamp.org/news/how-to-get-a-docker-container-ip-address-explained-with-examples/*, 22 June 2020.

[11] Docker Documentation. Run docker in the foreground. *https://docs.docker.com/engine/reference/run/#foreground*, 2020.

[12] Docker Documentation. Run docker: Cleanup after run. *https://docs.docker.com/engine/reference/run/#clean-up---rm*, 2020.

[13] Docker Documentation. Docker: Create & manage volumes. *https://docs.docker.com/storage/volumes/#create-and-manage-volumes*, 2020.

[14] Sofija Simic. Mysql docker container tutorial: How to set up & configure. *https://phoenixnap.com/kb/mysql-docker-container*, 10 February 2020.

[15] Docker Hub. Docker official images: Mysql. *https://hub.docker.com/_/mysql*.

[16] Janne Ruostemaa. How to install wordpress with docker. *https://upcloud.com/community/tutorials/wordpress-with-docker/*, 02 February 2020.

[17] Docker Hub. Docker hub: phpmyadmin. *https://hub.docker.com/r/phpmyadmin/phpmyadmin/*, 2020.

[18] Docker Documentation. Docker: Docker logs. *https://docs.docker.com/engine/reference/commandline/logs/*, 2020.

8 STUDENT HONESTY DECLARATION

Engaging in any cheating or dishonesty in any form of assessment, assignment, test orexamination or other WeThinkCode_ prescribed work is considered cheating and is grounds for disciplinary action. Plagiarism, which is to present work (or a portion of work) as your own when it is not, isconsidered cheating and is not accepted at WeThinkCode_.

An evaluator can flag one for plagiarism on one of the following grounds :

- The evaluator (marker) identifies that the student does not understand all or part of the work they have submitted.
- If all or part of the work presented is plagiarised ,i.e. copied from another source without reference.

Cheating in group projects

The main purpose for a group project is to give students the experience of working in ateam, by coming up with a solution to a problem together.

- Each member must be able to show which portion of the project they worked on.

- Failure to do so will result in the student being flagged for cheating which will begrounds for disciplinary action.
- This is to avoid single members doing the majority of the group project at the benefit of a member who is not contributing.
- In this way we are able to ensure fair assessment of each WTC_ student's competence.

Group projects can be approached in two ways.

1. Divide and conquer: This is usually preferred and advised when working on big projects. The project is divided into segments, in which each member of the group can accomplish. Once completed, the group will then integrate the segments to complete the project
2. One for all: This method is usually preferred and advised when a group is working on a small project. The group will work on the solution together from the start of the project until the end. This will require the members to move at a pace in which everyone in the team can keep up with.

NOTE: At the end of each group project, each member should have a general and basic understanding of the project and the solution found. This will include running,testing and explaining the solutions of the project.

DECLARATION

I hereby declare that the work submitted by me and/or my group members is:

- Original (not plagiarised)
- References listed
- Honest & in Good Faith
- Subject to WeThinkCode_policies

Sibonelo Nkosi
Username: sinkosi
Developer