Creating different Web servers and a Web Benchmark tool

Academic Area of Computer Engineering
CE-4303 – Operating Systems Principles
Rony Paniagua Chacón
Bryan Abarca Weber
Raúl Arias Quesada

March 2019

1 Introduction

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings [1].
Docker Engine creates simple tooling and a universal packaging approach that bundles up all application dependencies inside a container. Docker Engine enables containerized applications to run anywhere consistently on any infrastructure, solving "dependency hell" for developers and operations teams, and eliminating the "it works on my laptop!" problem [2].

2 Development environment

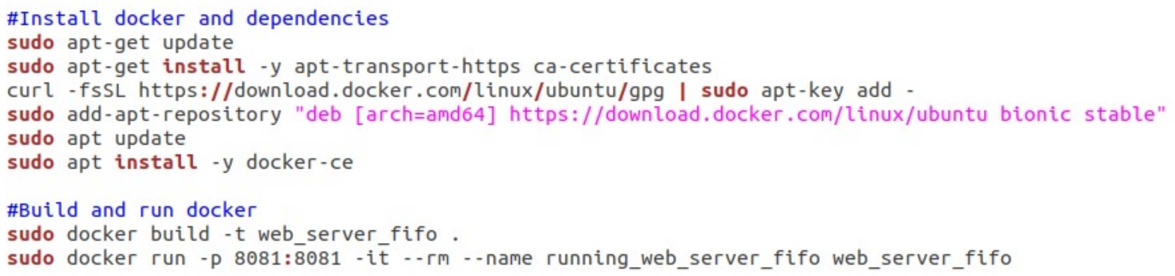
For the development of this project the tools were necessary:

1. Operating system with Linux distribution preferably Ubuntu.
2. C code editor, you can even use the text editor of the operating system, in our case we use the free editor Visual Studio Code.
3. Dockers: open source project used for the creation and testing of dockers, which use:
   - Docker Container: they are used to implement the services, each Docker contains the resources and necessary configuration of the environment that each of the services needs for its operation. Dockers bundle up all application dependencies inside container and enables containerized applications to run anywhere on any infrastructure [2].
     In other words, with Docker, it is possible to grab a portable runtime environment as an image, no installation necessary. Then, the build can include the base environment image right alongside your app code, ensuring that your app, its dependencies, and the runtime, all travel together [3]. These portable images are defined by something called a Dockerfile [3].
   - Dockerfile: defines what goes on in the environment inside your container. Access to resources like networking interfaces and disk drives is virtualized inside this environment, which is isolated from the rest of your system, so you need to map ports to the outside world, and be specific about what files you want to "copy in" to that environment. However, after doing that, you can expect that the build of your app defined in this Dockerfile behaves exactly the same wherever it runs [3].

3 Details of program design

3.1 Web server FIFO

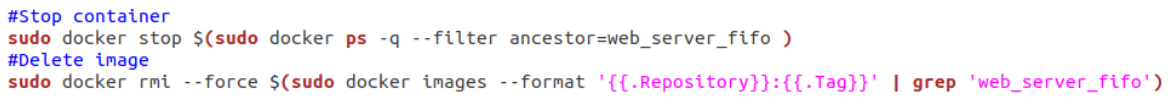
To install Docker Community Edition and its dependencies an automation script was created. In this script also is built and runned a docker, using the Dockerfile in the current directory, the content of this file is explained later.



```
#Install docker and dependencies
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
sudo apt update
sudo apt install -y docker-ce

#Build and run docker
sudo docker build -t web_server_fifo .
sudo docker run -p 8081:8081 -it --rm --name running_web_server_fifo web_server_fifo
```

Figure 1: Docker installer script.

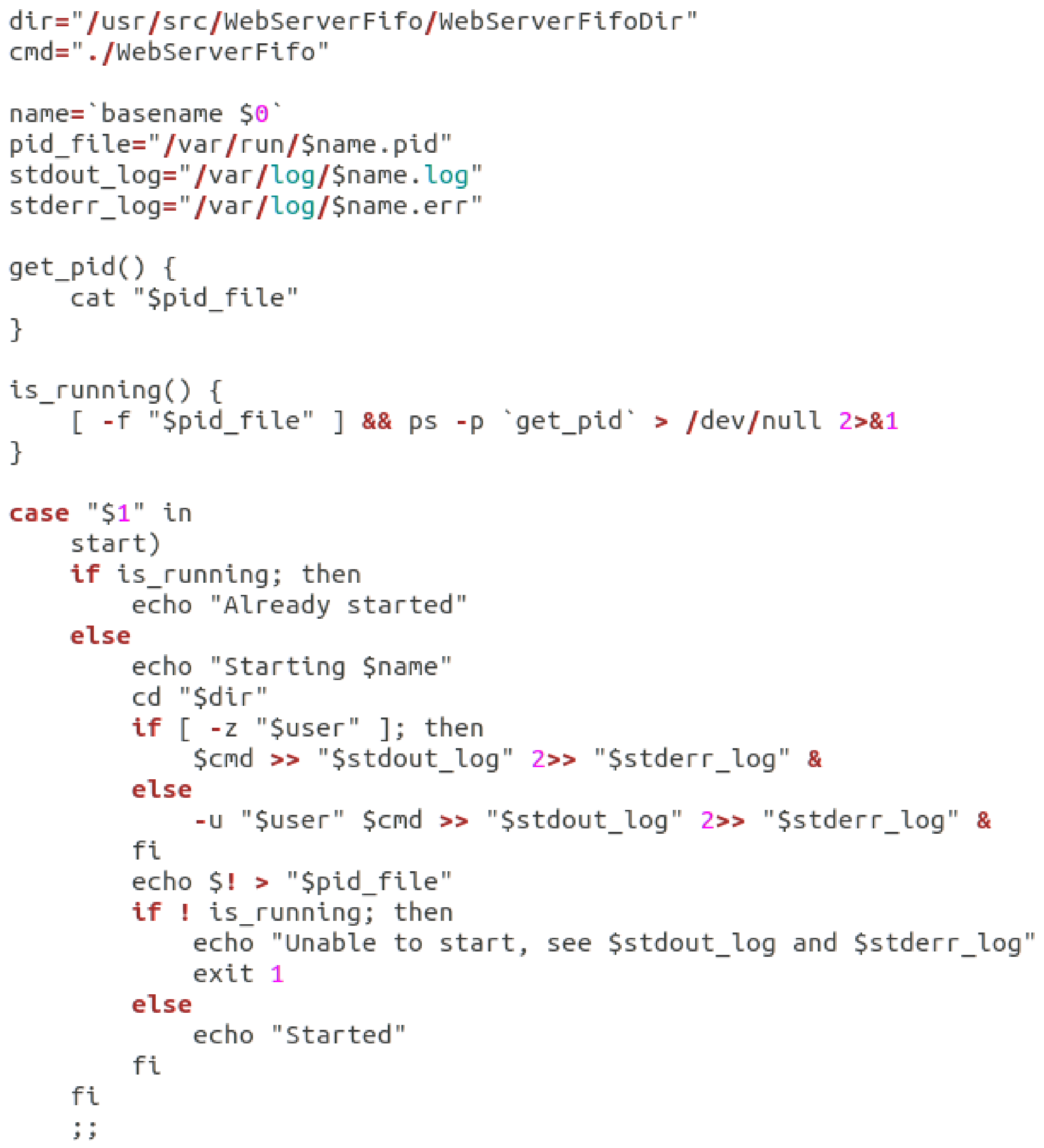
In contrast, to stop the docker container execution and uninstall it, the following script is executed.



```
#Stop container
sudo docker stop $(sudo docker ps -q --filter ancestor=web_server_fifo )
#Delete image
sudo docker rmi --force $(sudo docker images --format '{{.Repository}}:{{.Tag}}' | grep 'web_server_fifo')
```

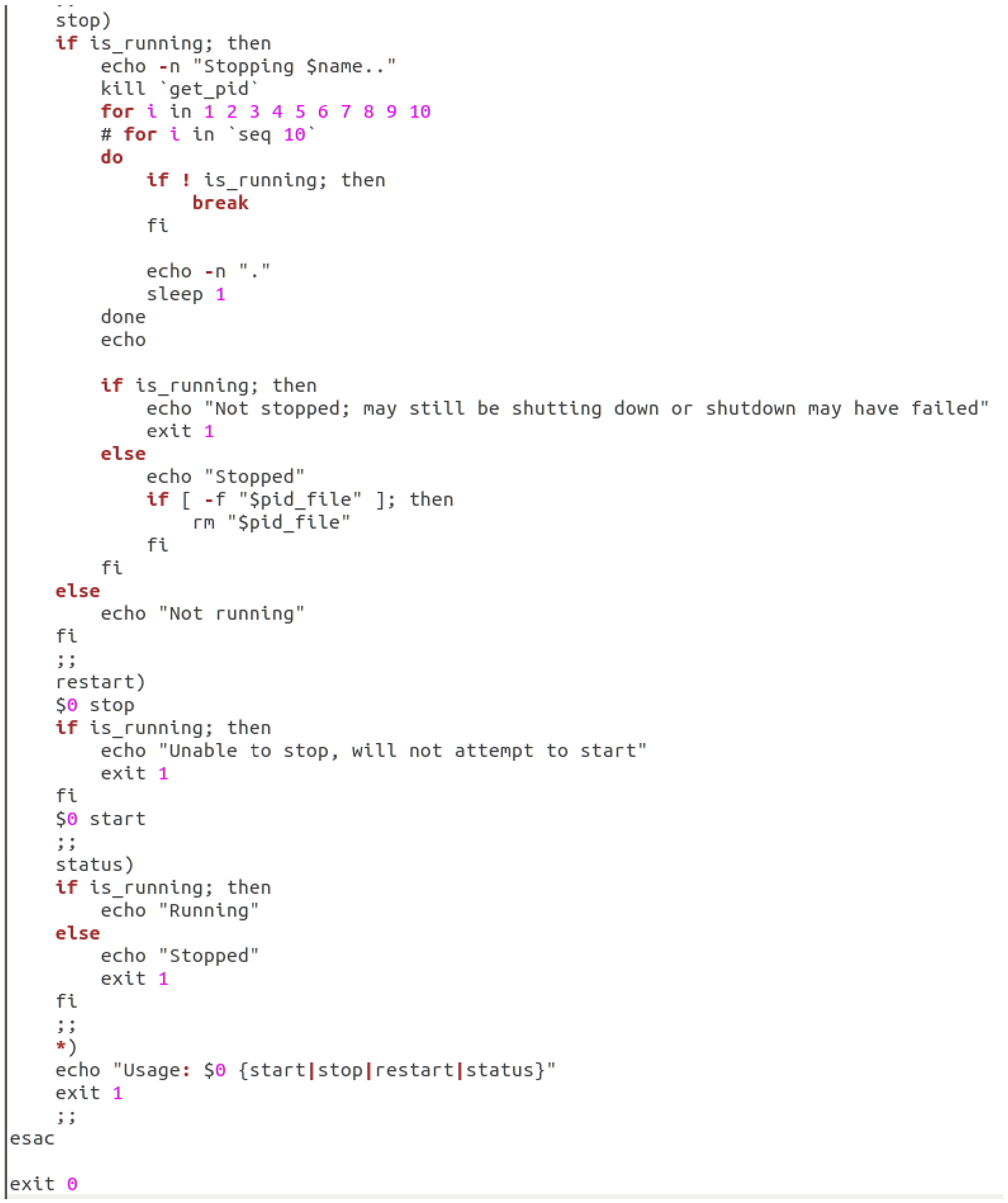
Figure 2: Docker uninstaller script.

To create a daemon service within the docker it is necessary to create a script that contains the operations of starting, stopping, restarting and viewing the state of the daemon, which has the content shown in the following figure.



```
dir="/usr/src/WebServerFifo/WebServerFifoDir"
cmd="./WebServerFifo"

name=`basename $0`
pid_file="/var/run/$name.pid"
stdout_log="/var/log/$name.log"
stderr_log="/var/log/$name.err"

get_pid() {
    cat "$pid_file"
}

is_running() {
    [ -f "$pid_file" ] && ps -p `get_pid` > /dev/null 2>&1
}

case "$1" in
    start)
    if is_running; then
        echo "Already started"
    else
        echo "Starting $name"
        cd "$dir"
        if [ -z "$user" ]; then
            $cmd >> "$stdout_log" 2>> "$stderr_log" &
        else
            -u "$user" $cmd >> "$stdout_log" 2>> "$stderr_log" &
        fi
        echo $! > "$pid_file"
        if ! is_running; then
            echo "Unable to start, see $stdout_log and $stderr_log"
            exit 1
        else
            echo "Started"
        fi
    fi
    ;;
```

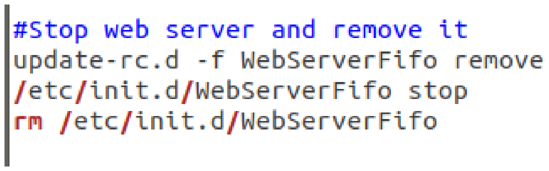
Figure 3: Web Server service installer script.



```
    ;;
    stop)
    if is_running; then
        echo -n "Stopping $name.."
        kill `get_pid`
        for i in 1 2 3 4 5 6 7 8 9 10
        # for i in `seq 10`
        do
            if ! is_running; then
                break
            fi

            echo -n "."
            sleep 1
        done
        echo

        if is_running; then
            echo "Not stopped; may still be shutting down or shutdown may have failed"
            exit 1
        else
            echo "Stopped"
            if [ -f "$pid_file" ]; then
                rm "$pid_file"
            fi
        fi
    else
        echo "Not running"
    fi
    ;;
    restart)
    $0 stop
    if is_running; then
        echo "Unable to stop, will not attempt to start"
        exit 1
    fi
    $0 start
    ;;
    status)
    if is_running; then
        echo "Running"
    else
        echo "Stopped"
        exit 1
    fi
    ;;
    *)
    echo "Usage: $0 {start|stop|restart|status}"
    exit 1
    ;;
esac

exit 0
```

Figure 4: Web Server service installer script.

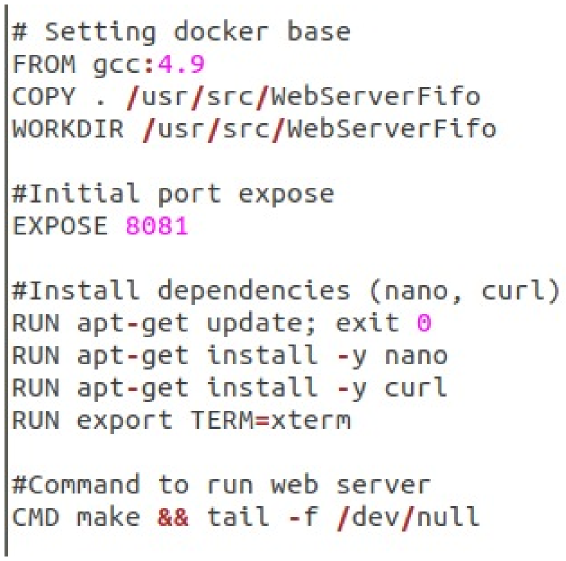
The following script is used to stop and remove the web server daemon service.



```
#Stop web server and remove it
update-rc.d -f WebServerFifo remove
/etc/init.d/WebServerFifo stop
rm /etc/init.d/WebServerFifo
```

Figure 5: Script to stop the web server daemon.

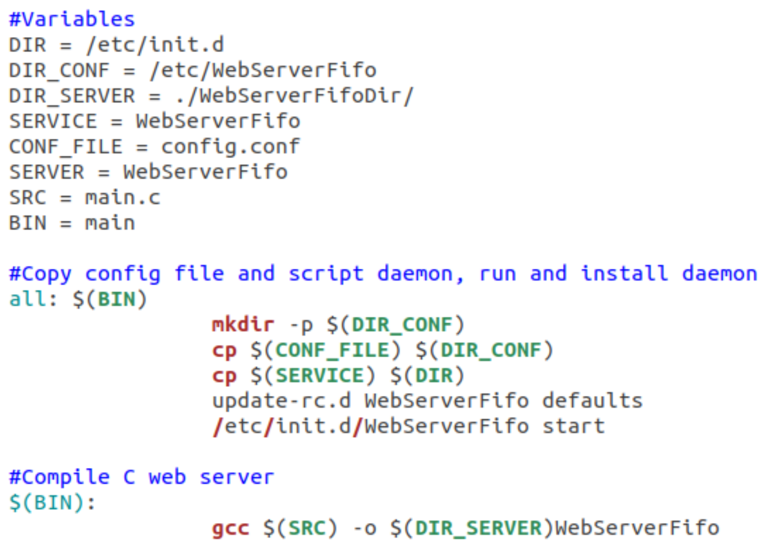
As mentioned, what goes on in the environment inside a container is defined by a Dockerfile. For instance in this project the Dockerfile in the figure 6, were used.



```
# Setting docker base
FROM gcc:4.9
COPY . /usr/src/WebServerFifo
WORKDIR /usr/src/WebServerFifo

#Initial port expose
EXPOSE 8081

#Install dependencies (nano, curl)
RUN apt-get update; exit 0
RUN apt-get install -y nano
RUN apt-get install -y curl
RUN export TERM=xterm

#Command to run web server
CMD make && tail -f /dev/null
```

Figure 6: Dockerfile.

In this file the environment inside the container is configured. First a gcc runtime environment is defined as the parent image, since the server application was implemented in C. The current directory contents are copied into the container at /usr/src/WebServerFifo, as well the working directory is set to the same directory.
The port 8081 is initially set available to the world outside this container. Then, all packages and dependencies needed are installed, and finally the Makefile is executed to perform the installation and configuration of the server when the container launches.
The Makefile content is presented in the figure 7.



```
#Variables
DIR = /etc/init.d
DIR_CONF = /etc/WebServerFifo
DIR_SERVER = ./WebServerFifoDir/
SERVICE = WebServerFifo
CONF_FILE = config.conf
SERVER = WebServerFifo
SRC = main.c
BIN = main

#Copy config file and script daemon, run and install daemon
all: $(BIN)
                mkdir -p $(DIR_CONF)
                cp $(CONF_FILE) $(DIR_CONF)
                cp $(SERVICE) $(DIR)
                update-rc.d WebServerFifo defaults
                /etc/init.d/WebServerFifo start

#Compile C web server
$(BIN):
                gcc $(SRC) -o $(DIR_SERVER)WebServerFifo
```

Figure 7: Makefile to install and configure the server.

The instructions in the Makefile are executed inside the docker container to install the web server. First the source code of this web server is compiled, generating a binary file located in /WebServerFifoDir in the current directory. Next the config file and the daemon script are copied to their respective folders and finally the daemon is installed and runned.

The web server has a main.c file that contains the functionality of it, based on [4]. With the following methods:

1. startServer: this method receives the port as input, it is responsible for lifting the web server, executing the socket and bind actions of the server, this using the sys / socket.h library.

2. SendResponse: It receives as input the request to answer, it is responsible for verifying the request and responding using HTTP-1.1 for it, it executes methods such as recv to read the request and write to respond.

3. main: Main method responsible for reading the values of configuration files, raise the server and keep listening and accepting connections using the other methods created.

4. getLogPath: Method that reads the configuration file and returns the path for the file where to write the logs, for it looks for the line that has the value of "LOGFILE", in case of not finding the value of the path it returns the value for defect /var/log/syslog. Use methods such as fopen to open the file, fgets to read lines from the file, and sscanf to find the correct line.

5. getPort: Method that reads the configuration file and returns the port to use, for it looks for the line that has the value of "PORT", in case of not finding the value of the port it returns the default value 8081. It uses methods as fopen to open the file, fgets to read lines from the file and sscanf to find the correct line.

6. write: Receive the data to write and the path of the file to write, in addition to writing the required message write the time of the system to have a better log of the activity. Use the time.h library to get the time and methods such as open to open the file and write to write to it.

3.2 Benchmark tool

The benchmark tool to develop is an application that will take the console parameters (<machine> <port> <file>r <N-threads> <N-cycles>) and interpret them to create N number of threads with the mylpthread library where each one of these threads will open a connection socket to the server of the specified machine, each of these sockets will make the corresponding request the specified number of times. A general diagram is presented in the figure 8 of how the benchmark tool works

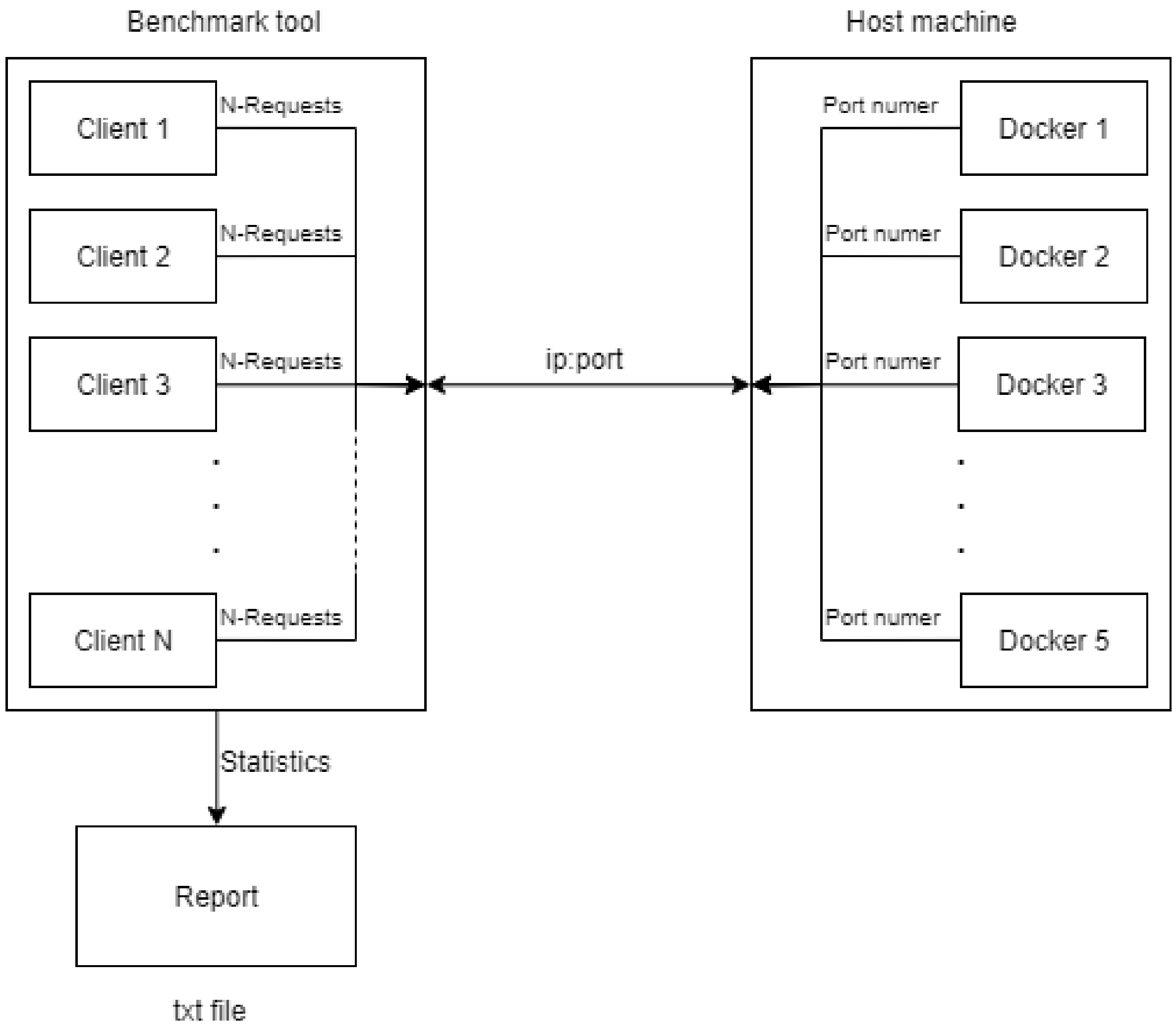
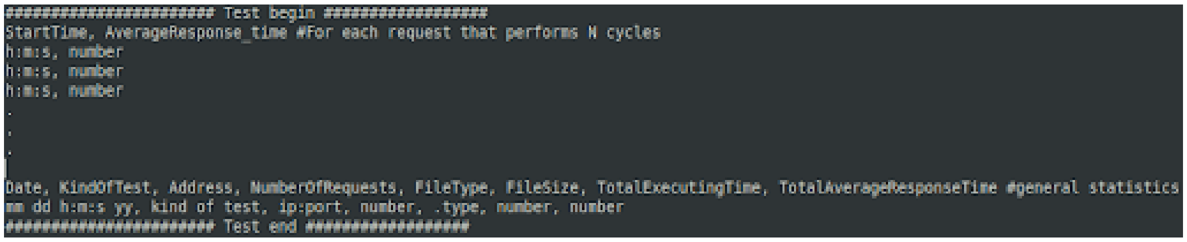


Figure 8: Benchmarck diagram.

Each thread created for the requests will be in charge of storing the information of the corresponding times, that is, the average response time of the server in each request of the N cycles carried out. The time at which each of the cycles began to run will also be saved.
At the end of the test, general statistics of the result will be saved, such as: the date on which it was performed, the type of test (depending on the type of server), the IP address and port of the server, the number of requests made, the type of file that was requested, the size of the file, the total seconds that the test took and the average response time of all requests made. Figure 9 shows an example of the format that the test report will have.



```
####################### Test begin ##################
StartTime, AverageResponse_time #For each request that performs N cycles
h:m:s, number
h:m:s, number
h:m:s, number
.
.
.

Date, KindOfTest, Address, NumberOfRequests, FileType, FileSize, TotalExecutingTime, TotalAverageResponseTime #general statistics
mm dd h:m:s yy, kind of test, ip:port, number, .type, number, number
####################### Test end ##################
```

Figure 9: Benchmarck report example.

4 Instructions of how to use the programs

4.1 Installation and use of fifo web server

4.1.1 Requirements

- Work with Ubuntu (tested on a azure VM with Ubuntu 18.04).

4.1.2 Creation and execution of the docker

To create the docker it is only necessary to execute command: sh InstallDockerFifo.sh in the path where the delivered files are located, this is responsible for installing docker in the machine, do the docker build and execute it. It is necessary to add nano and curl to the docker, for that reason the installation commands are added to the docker, but when executing them some messages in red of some things that are not found are shown, however, this does not interfere in the correct functioning of the docker.

4.1.3 Docker execution test

When executing the previous command, the docker remains running. By default it is ready to run on port 8080, so to do tests it would only be necessary to go to a browser and make requests. Currently there are several test files in the directory of the web server (/usr/src/WebServerFifo/WebServerFifoDir), so the possible requests would be:

http://server:8081/test.png
http://server:8081/test.pdf
http://server:8081/test.html
http://server:8081/test.bin
http://server:8081/test.mp4

4.1.4 Modify web server config and operations with the daemon

To enter the running docker you must open another terminal and execute the command:

sudo docker exec -it running_web_server_fifo bash

To perform the requested operations (start, stop, restart, view status) of the web server, as well as modify the configuration file, the following commands must be executed:

To perform actions on the daemon:

Start daemon: /etc/init.d/WebServerFifo start
Stop daemon: /etc/init.d/WebServerFifo stop
Restart daemon: /etc/init.d/WebServerFifo restart
Daemon status: /etc/init.d/WebServerFifo status

To modify the configuration file

–nano /etc/WebServerFifo/config.conf

4.1.5 Uninstall docker

To uninstall the docker it is necessary to execute the command: sh RemoveDockerFifo.sh in the path where the delivered files are located

5 Student activity log

Table 1: Bryan Abarca Activity Log

Activity	**Duration**
Reading and analysis of the project specification.	1 h
Research on the operation and use of docker containers.	2 h
Implementation of the docker container and and adjustments in the makefile.	6 h
Documentation.	3 h
Total	12 h

Table 2: Raúl Arias Activity Log

Activity	**Duration**
Reading and analysis of the project specification.	1 h
Error correction of FIFO web server (logs, makefile, uninstall file).	4 h
Research on the operation and use of docker containers.	2 h
Support in the creation of the docker.	2 h
Implementation of the daemon for SysV.	2 h
Documentation.	2 h
Total	13 h

Table 3: Rony Paniagua Activity Log

Activity	**Duration**
Reading and analysis of the project specification.	1 h
High level design of the benchmark.	3 h
Research on the operation and use of docker containers.	2 h
Implementation of the installation and destruction script of the docker container.	2 h
Documentation.	3 h
Total	11 h

6 Project final status

So far we have the successful implementation of the docker that contains the fifo web server which works correctly without errors detected.

7 Conclusions

The use of dockers presents a series of advantages such as the compatibility that allows to execute applications independently of the platform in which they are being executed, besides it allows to realize a standardization and simplicity of configurations, all the previous allows the deploy of applications to be much easier and faster regardless of the platform used.

8 Suggestions

It is recommended to start with images of dockers that have installed some of the components that are needed, for example the start of the docker from gcc avoids having to add the commands necessary for the installation of gcc

9 References

References

[1] Docker. [Online]. Available in: https://www.docker.com/resources/what-container

[2] Docker. [Online]. Available in: https://www.docker.com/products/docker-engine

[3] docker docs. Get started, part 2: Containers. [Online]. Available in: https://docs.docker.com/get-started/part2/

[4] Rastogi, A. (2010). A Very Simple HTTP Server writen in C. [Online] Abhijeet's Blog. Available in: https://blog.abhijeetr.com/2010/04/very- simple-http-server-writen-in-c.html.