



Al-Najah National University
Computer Network and Information Security
Network Administration Lab

Docker

Instructor:Dr. Ahmed Awad

Dima Bshara
Mohammed Adnan

1 Abstract

This experiment explains Linux containers and differentiates between a container and a VM. Also, list solutions that could use to implement containers and how to install and configure docker. In the end, we build our container image.

2 Introduction

If we almost look at how Virtual Machines are built over the physical hardware, there is a layer of Hypervisor, which sits between physical hardware and operating systems. In a broader view, Hypervisor is used to virtualize that hardware which is then configuring in the way a user wants it.

Docker is a set of platforms as service products that use OS-level virtualization to deliver software in packages called containers. Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. Using Docker, you can quickly deploy and scale applications into any environment and know your code will run Let us look at the component view of how virtual machines and docker are implemented as shown in figure 1. Also figure 2 shows the main different between docker and VM.

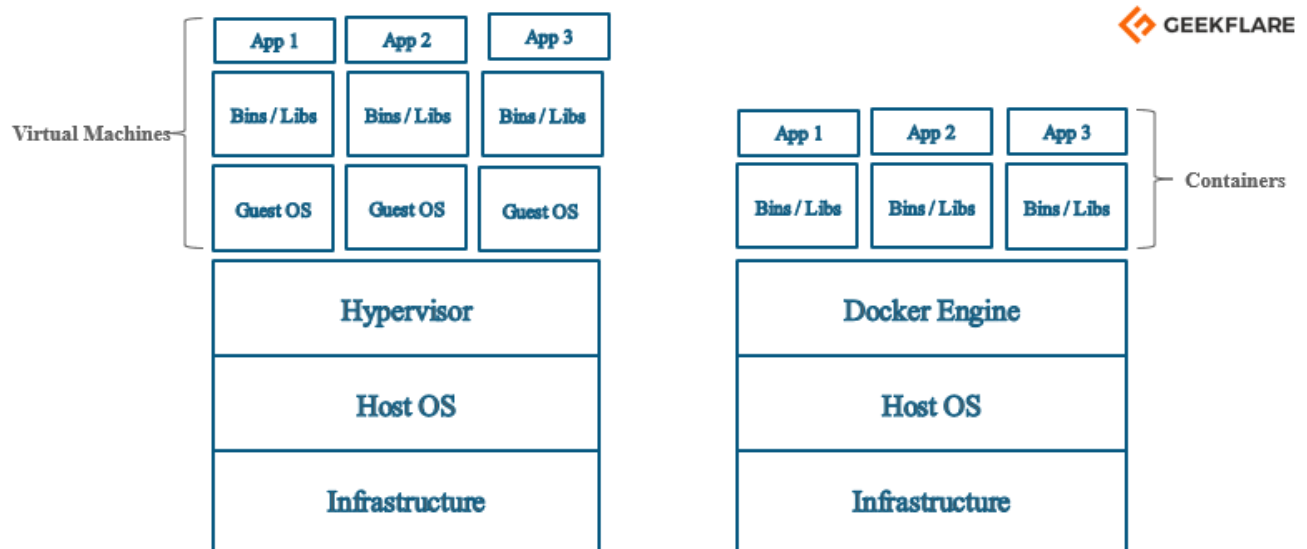


Figure 1: docker vs vm.

Virtual Machines	Docker
Each VM runs its own OS	All containers share the same Kernel of the host
Boot up time is in minutes	Containers instantiate in seconds
VMs snapshots are used sparingly	Images are built incrementally on top of another like layers. Lots of images/snapshots
Not effective diffs. Not version controlled	Images can be diffed and can be version controlled. Dockerhub is like GITHUB
Cannot run more than couple of VMs on an average laptop	Can run many Docker containers in a laptop.
Only one VM can be started from one set of VMX and VMDK files	Multiple Docker containers can be started from one Docker image

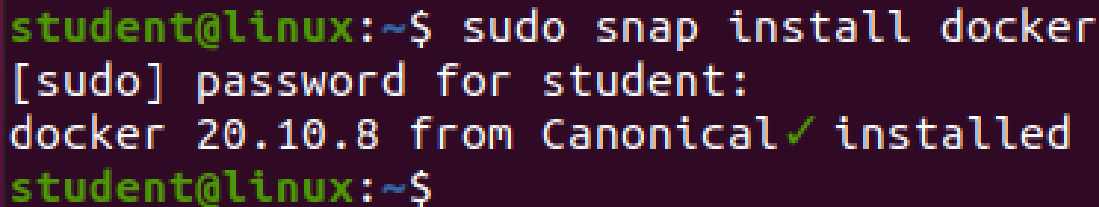
Figure 2: docker vs vm.

Unlike virtual machines where hypervisor divides physical hardware into parts, Containers are like normal operating system processes. Now the question is, if containers are like normal processes then how come there are isolated from other processes. This is where namespaces come into the picture. A namespace is an advanced concept in Linux where each namespace has its isolated resources without actual partitioning of the underlying hardware. Namespaces are used to virtualize the underlying operating system. Since containers are nothing other than OS processes, This is the reason why lifting a container takes seconds and lifting a virtual machine takes minutes.

3 Procedure

3.1 Install Docker

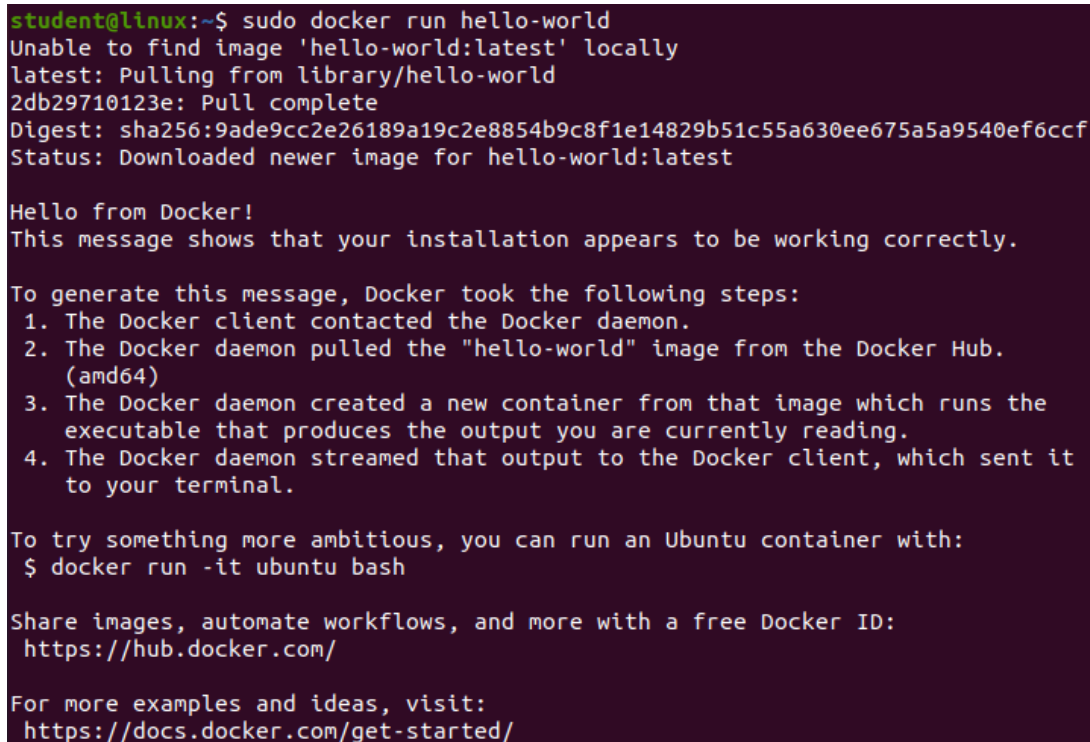
1. At first, we use the snap store to install docker using the following command to install Docker: `sudo snap install docker` as figure 3 show.



```
student@linux:~$ sudo snap install docker
[sudo] password for student:
docker 20.10.8 from Canonical✓ installed
student@linux:~$
```

Figure 3: Install Docker.

2. Then verify that docker installed successfully using the following command: `Sudo docker run hello-world`, as figure 4 shows, note: when docker doesn't find the image `hello-world`, docker will pull it.



```
student@linux:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:9ade9cc2e26189a19c2e8854b9c8f1e14829b51c55a630ee675a5a9540ef6ccf
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.
   (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/
```

Figure 4: verify Docker.

3. We set the current user account to use docker without root permissions, firsts, create a new group with the name docker, then add the user to the supplementary group use only with the -G option, final attempts to log in to the group, as figure 5 show.

```
student@linux:~$ sudo groupadd docker
student@linux:~$ sudo usermod -aG docker $USER
student@linux:~$ newgrp docker
student@linux:~$
```

Figure 5: Set current user account to use docker without root permissions

4. Then verify that user have permission to use docker without root permissions, as figure 6 show.

```
student@linux:~$ 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/
```

Figure 6: verify permission.

3.2 Download images

Docker hub is the main source of docker images, these are prebuilt containers that we can download and use for creating containers we need.

1. Download ubuntu using command **docker pull ubuntu**, this will download the latest version of that image, as figure 7 show.

```
student@linux:~$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
f3ef4ff62e0d: Pull complete
Digest: sha256:a0d9e826ab87bd665cfc640598a871b748b4b70a01a4f3d174d4fb02adad07a9
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
student@linux:~$
```

Figure 7: verify permission.

2. After that, we download ubuntu with another version that specifies it with version number (tag), as figure 8 shows.

```
student@linux:~$ docker pull ubuntu:18.04
18.04: Pulling from library/ubuntu
284055322776: Pull complete
Digest: sha256:bfb4cabd667790ead5c95d9fe341937f0c21118fa79bc768d51c5da9d1dbe917
Status: Downloaded newer image for ubuntu:18.04
docker.io/library/ubuntu:18.04
student@linux:~$
```

Figure 8: verify permission.

3. Then we check the images have been downloaded use the command: **docker images**, as figure 9 show.

```
student@linux:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	597ce1600cf4	11 days ago	72.8MB
ubuntu	18.04	5a214d77f5d7	11 days ago	63.1MB
hello-world	latest	feb5d9fea6a5	2 weeks ago	13.3kB

```
student@linux:~$
```

Figure 9: List the images

3.3 Container management

1. We create a container from an image using the command: **docker run -dt --name ubuntu01 ubuntu**, then we use command **docker ps** to verify container is running notice that ubuntu01 is listed as running as figure 10 show.

```
student@linux:~$ docker run -dt --name ubuntu01 ubuntu
386ca5780001491048a28b2968932996119533e03a62333ef183c502d27b6b15
student@linux:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
386ca5780001	ubuntu	"bash"	29 seconds ago	Up 28 seconds		ubuntu01

```
student@linux:~$
```

Figure 10: Create and run Container.

Note:

-d: detached so that we don't enter the command line of the container immediately which will be a problem since we did not specify a shell to use meaning we will be stuck on an empty screen.

-t: attach a terminal for the container. Without this command the container will stop immediately after starting. This is because containers are supposed to run apps inside they exist for only that. So if a containers' app exits the containers gets destroyed and if no app is running the container stops after starting.

2. Then we create another container using the same image but in a slightly different way by changing the name, and check its status using the command ps, after that, we check all container status using **docker ps -a** as figure 11 shows.

```
student@linux:~$ docker run -d --name ubuntu02 ubuntu
19d03cd01d4f07a0e8eb983ac15901a6b0eb04b246747c5f290c031481f8baf4
student@linux:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
386ca5780001	ubuntu	"bash"	2 minutes ago	Up 2 minutes		ubuntu01

```
student@linux:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
19d03cd01d4f	ubuntu	"bash"	3 minutes ago	Exited (0) 3 minutes ago		ubuntu02
386ca5780001	ubuntu	"bash"	4 minutes ago	Up 4 minutes		ubuntu01
4408d880bfa2	hello-world	"/hello"	16 minutes ago	Exited (0) 16 minutes ago		wizardly_
41f886857a90	hello-world	"/hello"	20 minutes ago	Exited (0) 20 minutes ago		upbeat_li

Figure 11: Create, run container and list all the container status .

Question: What is the status of ubuntu02? *Exited*, because we didn't run any service for now.

3. After that, we execute a command inside the container as shown 12, first commands show the current directory, the second one is for list all files in dir for home dir, so as we see that there's no file before creation test.txt file under the home directory.

```
student@linux:~$ docker exec ubuntu01 pwd
/
student@linux:~$ docker exec ubuntu01 pwd
/home/student
student@linux:~$ docker exec ubuntu01 ls /home
student@linux:~$ docker exec ubuntu01 touch /home/test.txt
student@linux:~$ docker exec ubuntu01 ls /home
test.txt
```

Figure 12: Create file in container under home dir .

4. To enter the shell of the containers, we have two ways; as shown in figure 13, the first way is the first command (-t) will attach a terminal (-i) will enable standard input so that we can input commands. And the other one in the second command will need (-i) in the running to be interactive.

```
student@linux:~$ docker exec -it ubuntu01 sh
# ls
bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys
# ls /home
test.txt
# exit
student@linux:~$ docker attach ubuntu01
root@386ca5780001:/# exit
```

Figure 13: Enter the shell of container .

5. Then we stopped the container and made sure that it stopped by using the ps command. After that, we start it again and verify that it runs, also by using the ps command, figure 14 shows those steps.

```
(mohammed@mohammed)-[~/containersdir]
$ docker ps
CONTAINER ID   IMAGE          COMMAND         CREATED        STATUS        PORTS        NAMES
ba2bd5363aad   ubuntu:20.04   "bash"         12 minutes ago Up 5 seconds   ubuntu40

(mohammed@mohammed)-[~/containersdir]
$ docker stop ubuntu40
ubuntu40

(mohammed@mohammed)-[~/containersdir]
$ docker ps
CONTAINER ID   IMAGE          COMMAND         CREATED        STATUS        PORTS        NAMES

(mohammed@mohammed)-[~/containersdir]
$ docker start ubuntu40
ubuntu40

(mohammed@mohammed)-[~/containersdir]
$ docker ps
CONTAINER ID   IMAGE          COMMAND         CREATED        STATUS        PORTS        NAMES
ba2bd5363aad   ubuntu:20.04   "bash"         13 minutes ago Up 7 seconds   ubuntu40
```

Figure 14: start and stop container.

6. We stop the container and remove it as figure 15 shows, then we create it again and run it to check if the file that we created is still under the home dir or not, so as we conclude that the file will remove directly when the container deleted because the containers are volatile as shown in Figures 16 and 17.

```
student@linux:~$ docker stop ubuntu01
ubuntu01
student@linux:~$ docker rm ubuntu01
ubuntu01
student@linux:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAM
19d03cd01d4f	ubuntu	"bash"	19 minutes ago	Exited (0) 19 minutes ago		ubu
4408d880bfa2	hello-world	"/hello"	33 minutes ago	Exited (0) 33 minutes ago		wiz
41f886857a90	hello-world	"/hello"	37 minutes ago	Exited (0) 37 minutes ago		upb

```
student@linux:~$
```

Figure 15: stop and remove container.

```
student@linux:~$ docker run -dt --name ubuntu01 ubuntu
3b19c0d0c769db817a1bdafe21671a4bc986ef1f7d04d42dafcda0d7357c2158
student@linux:~$ docker attach ubuntu01
```

Figure 16: create it again .

```
student@linux:~$ docker exec -it ubuntu01 sh
# ls /hom
ls: cannot access '/hom': No such file or directory
# ls /home
# ls
```

Figure 17: list the file under home dir.

3.4 Persistent Storage Containers

1. Containers are volatile, and to make their storage persistent, we need to map a directory inside the container to a directory inside our host machine. First, we create a directory in /home directory named containerdir, then we create a container with directory mapping as shown in figure 18. After that, we create a file in the home directory of this container, as figure 19 shows. In the end, we stop and remove the container, and we made sure that the contents of the containerdir directory existing as shown in figure 20.

```
student@linux:~$ sudo mkdir /home/student/containerdir
student@linux:~$ docker run -dt -v /home/student/containerdir:/home/ ubuntu
059b8223bf1105ac17069b641e4719bc3a2282a23cd0bdccfa174d4da1b0650e
student@linux:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
059b8223bf11	ubuntu	"bash"	8 seconds ago	Up 6 seconds
19d03cd01d4f	ubuntu	"bash"	39 minutes ago	Exited (0) 39 minutes

Figure 18: create dir and mapped it with container.

```
student@linux:~$ docker exec -ti beautiful_benz sh
#
student@linux:~$ docker attach beautiful_benz
root@059b8223bf11:/# sudo ls
^C
student@linux:~$ docker exec -ti beautiful_benz sh
# touch /home/testfile
# exit
student@linux:~$
```

Figure 19: create file inside container.

```
student@linux:~$ docker stop beautiful_benz
beautiful_benz
student@linux:~$ docker rm beautiful_benz
beautiful_benz
student@linux:~$ ls /home/student/containerdir/
testfile
student@linux:~$
```

Figure 20: delete container and made sure of contents of containerdir.

2. If we have running services like HTTP on our container, then only be accessible by the host machine. At first, we run the web server as shown in figure 21, then we enter the hostname -I in the shell of the webserver to get the IP for the HTTP server. Now on our host, access the site using curl HTTP://[containerIP], this should display the HTML code of the start page of the server, as figure 22 shows. This address is a host-only network address, that can be accessed only by the host.

```
student@linux:~$ docker run -dt --name webserver httpd
Unable to find image 'httpd:latest' locally
latest: Pulling from library/httpd
07aded7c29c6: Pull complete
05bb40c8f148: Pull complete
0827b74117da: Pull complete
197d463eb419: Pull complete
a6714064df9a: Pull complete
Digest: sha256:383138dcfd7bde29108500d38cd7f021ff9513057d446c01661cf1b0df219449
Status: Downloaded newer image for httpd:latest
625b6bfbfc267e85b81066b6e4074d0eb204be5033b75ab3c761de7ded2212c6
student@linux:~$
```

Figure 21: Run http server.

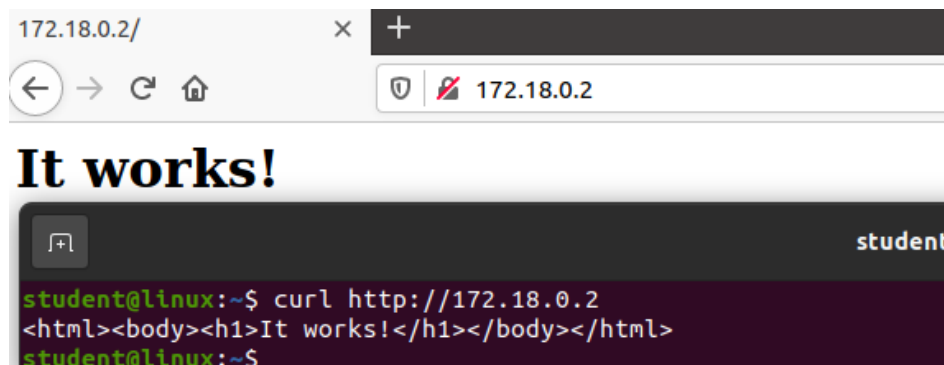


Figure 22: curl command.

3. As we see that this is not practical, we need other hosts able to access the site. We create a new container with port mapping enabled as shown in figure 23, as the figure shows that we mapped our website to port 80 so anyone wont to access our website using HTTP, it will direct it to our website in port 80.

```
student@linux:~$ docker run -dt -p 80:80 --name webserver httpd
3e8252c79b70449b32db005b0e600bea0fa565074c09b9e6b9c75c57d08c0626
student@linux:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS
3e8252c79b70   httpd     "httpd-foreground"      13 seconds ago Up 12 seconds  0.0.0.0:80->80/tcp, :::80->80/tcp
student@linux:~$
```

Figure 23: Mapping port.

3.5 To do

1. We create Container 1, which is based on ubuntu 20.04 with a web server running. The website is persistent with the index.html as shown in figure 24. After installing the apache server in container 1 we made sure that the index.html is persistent in the location that we put. Figure 25 shows those steps. In the end, we edit the index.html file, and we add our group name to it as shown in figure 26.

```
(mohammed@mohammed)-[~/contsinerdir]
$ docker run -dit -p 8080:8080 --name ubuntu400 -v /home/mohammed/contsinerdir:/var/www/html/ ubuntu:20.04
89ac8c044b0059b53c0c081c8bc0510b9ee220be9a24298207dba10da3675fbe

(mohammed@mohammed)-[~/contsinerdir]
$ docker attach ubuntu400
root@89ac8c044b00:/# apt update
Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
```

Figure 24: Running container with persistent.

```
(mohammed@mohammed)-[~]
$ docker attach ubuntu20
root@cca5ab432062:/# ls /var/www/html/
index.html
root@cca5ab432062:/# exit

(mohammed@mohammed)-[~]
$ ls contsinerdir
index.html

(mohammed@mohammed)-[~]
$
```

Figure 25: Made sure for index.html.

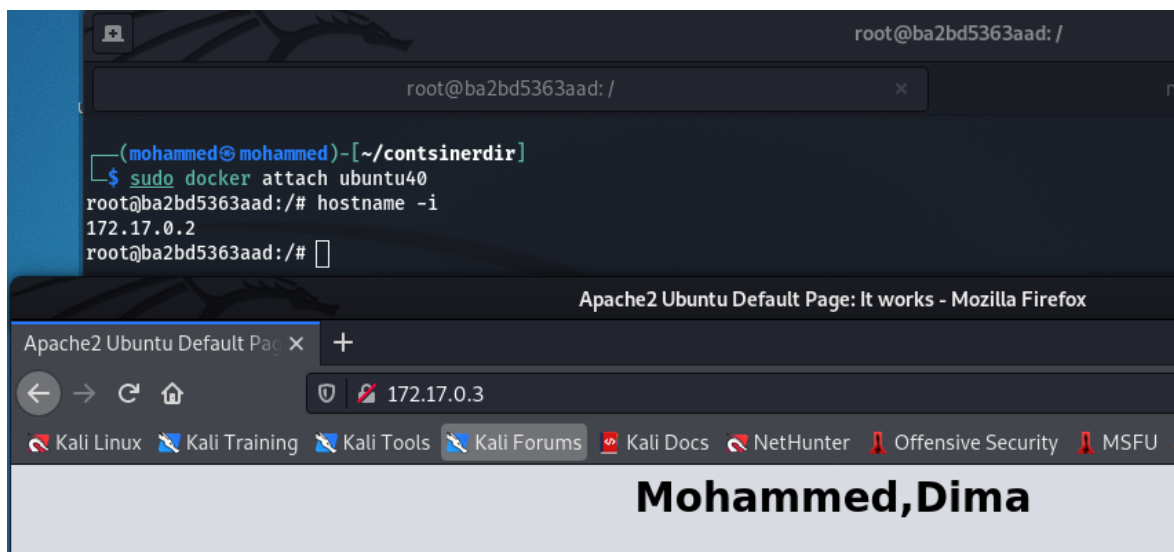


Figure 26: names of our group members.

2. Then we create Container 2 based on httpd with all default settings. This also is accessible from all other machines, as shown in figure 27.

```
(mohammed@ mohammed)-[~/contsinerdir/webserver]
$ docker run -dit --name webservers -v /home/mohammed/contsinerdir/webserver:/usr/local/apache2 httpd ba
5494609ea73aacdc5e0c9afde5a171c353ddc07735a5d4edeeda8fd03a88b939

(mohammed@ mohammed)-[~/contsinerdir/webserver]
$ docker exec -ti webservers sh
# ls
#

(mohammed@ mohammed)-[~/contsinerdir/webserver]
$ ls

(mohammed@ mohammed)-[~/contsinerdir/webserver]
$ docker exec -ti webservers sh
# pwd
/usr/local/apache2
# ls
#
```

Figure 27: Container based on httpd.

4 Conclusion

Using Docker lets you ship code faster, standardize application operations, seamlessly move code, and save money by improving resource utilization. With Docker, you get a single object that can reliably run anywhere. Docker's simple and straightforward syntax gives you full control. Wide adoption means there's a robust ecosystem of tools and off-the-shelf applications that are ready to use with Docker. Prime difference between Containers and Virtual Machine is the virtualization of Operating System and Hardware respectively. While using Virtual machine each virtual machine has its own underlying operating system whereas, While using containers, Each container runs on same underlying operating system instance, While underlying OS is the same containers can still have different OS environment in their respective namespace.

5 Reference

<https://www.docker.com/resources/what-container>
<https://aws.amazon.com/docker/>
[https://en.wikipedia.org/wiki/Docker\(*software*\)](https://en.wikipedia.org/wiki/Docker_(software))
<https://docs.docker.com/get-docker/>