Docker Exercise

Lab1: Docker basics Exercise 1: Install docker

1. Log in to your VM.
2. Start terminal and elevate your privileges to root.

```
andrijanasharkoska@Andrijana:~$ sudo -i
[sudo] password for andrijanasharkoska:
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.15.90.1-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage


This message is shown once a day. To disable it please create the
/root/.hushlogin file.
root@Andrijana:~# whoami
root
root@Andrijana:~#
```

3. Run yum install docker.

- Here I had to run the command **apt install docker**

```
root@Andrijana:~# apt install docker
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  wmdocker
The following NEW packages will be installed:
  docker wmdocker
0 upgraded, 2 newly installed, 0 to remove and 30 not upgraded.
Need to get 14.3 kB of archives.
After this operation, 58.4 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://archive.ubuntu.com/ubuntu jammy/universe amd64 wmdocker amd64 1.5-2 [13.0 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy/universe amd64 docker all 1.5-2 [1316 B]
Fetched 14.3 kB in 0s (33.8 kB/s)
Selecting previously unselected package wmdocker.
(Reading database ... 42527 files and directories currently installed.)
Preparing to unpack .../wmdocker_1.5-2_amd64.deb ...
Unpacking wmdocker (1.5-2) ...
Selecting previously unselected package docker.
Preparing to unpack .../archives/docker_1.5-2_all.deb ...
Unpacking docker (1.5-2) ...
Setting up wmdocker (1.5-2) ...
Setting up docker (1.5-2) ...
Processing triggers for man-db (2.10.2-1) ...
root@Andrijana:~#
```

- I've installed version 20.10.12

4. After installation is finished, start docker by running this command systemctl start docker.

```
Run 'docker swarm COMMAND --help' for more information on a command.
root@Andrijana:~# service docker start
root@Andrijana:~# 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/
```

- Here I had some issues with the ***systemctl start docker*** command, so I had to use ***service docker start*** to start Docker successfully.
- I've also run a Hello World service to see whether the installation works correctly. By the output, we can see it's working.

5. Also enable docker service automatic start with command systemctl enable docker.

```
root@Andrijana:~# systemctl enable docker
root@Andrijana:~#
```

6. Run docker version to see installed version.

```
root@Andrijana:~# docker -v
Docker version 20.10.21, build 20.10.21-0ubuntu1~22.04.2
root@Andrijana:~#
```

7. Run docker help to see list of available commands.

```
root@Andrijana:~# docker help

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default "/root/.docker")
  -c, --context string     Name of the context to use to connect to the daemon (overrides DOCKER_HOST env var and default context set with "docker context use")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level ("debug"|"info"|"warn"|"error"|"fatal") (default "info")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default "/root/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default "/root/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default "/root/.docker/key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

Management Commands:
  builder     Manage builds
  buildx*     Docker Buildx (Docker Inc., v0.10.3)
  compose*    Docker Compose (Docker Inc., v2.15.1)
  config      Manage Docker configs
  container   Manage containers
  context     Manage contexts
  dev*        Docker Dev Environments (Docker Inc., v0.1.0)
  extension*  Manages Docker extensions (Docker Inc., v0.2.18)
  image       Manage images
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  sbom*       View the packaged-based Software Bill Of Materials (SBOM) for an image (Anchore Inc., 0.6.0)
  scan*       Docker Scan (Docker Inc., v0.25.0)
  scout*      Command line tool for Docker Scout (Docker Inc., v0.6.0)
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
```

8. Search for a command (switch) that will show system-wide information for your instance of docker.

```
root@Andrijana:~# docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          3         3         1.1GB     0B (0%)
Containers      3         0         3.074MB   3.074MB (100%)
Local Volumes   1         1         2.598GB   0B (0%)
Build Cache     0         0         0B        0B
root@Andrijana:~#
```

- We run docker system df to display system-wide information of the docker instance

9. Test it by running docker and the command you found .

- I used **docker system df** command

10. From the output try to find where the information of number of containers and images is.

```
root@Andrijana:~# doc
TYPE            TOTAL
Images          3
Containers      3
```

11. Also try to find whether this docker is part of a swarm. Hint: Use Linux grep filtering if the output of this command is too verbose for you.

```
Docker version 20.10.21, build 20.10.21-0ubuntu1~22.04.2
root@Andrijana:~# docker swarm

Usage:  docker swarm COMMAND

Manage Swarm

Commands:
  ca          Display and rotate the root CA
  init        Initialize a swarm
  join        Join a swarm as a node and/or manager
  join-token  Manage join tokens
  leave       Leave the swarm
  unlock      Unlock swarm
  unlock-key  Manage the unlock key
  update      Update the swarm

Run 'docker swarm COMMAND --help' for more information on a command.
root@Andrijana:~#
```

- I used the docker swarm command to access the information.

Docker Swarm is a clustering and scheduling tool for Docker containers. With Swarm, IT administrators and developers can establish and manage a cluster of Docker nodes as a single virtual system. Swarm mode also exists natively for Docker Engine, the layer between the OS and container images.

Lab2: Creating images Excercise1: Build a simple image

1. Create a Docker container that executes a simple bash script. Go to your home directory and run mkdir test. Run cd test.
2. Create a simple script. Run vi test.sh.

```
root@Andrijana:~# touch test.sh
```

- I am using the nano editor

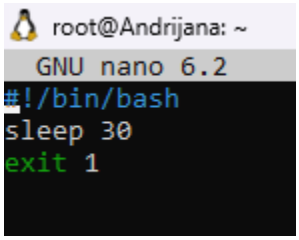
3. Write the following in your script file:

```
#!/bin/bash
sleep 30
exit 1
```

```
root@Andrijana:~# cat test.sh
#!/bin/bash
sleep 30
exit 1
```

- Here's the code output with the cat command

4. Save the file. In vi editor press :wq.

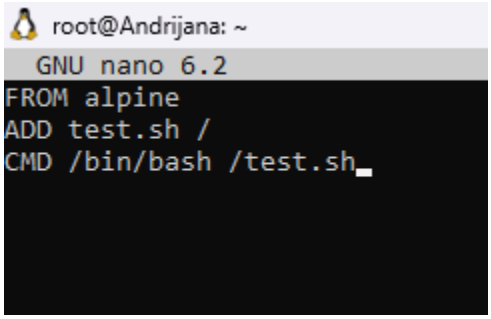


- I've used the nano editor and saved the file with Ctrl + O or Ctrl + X since the changes will be saved upon exit

5. Create a docker file. Run vi Dockerfile.

```
root@Andrijana:~# touch Dockerfile
```

6. Write the following in our Dockerfile:
FROM alpine
ADD test.sh /
CMD /bin/bash /test.sh



7. Save your Dockerfile.

8. Build your image. Run docker build –t my-image1 ./

```
root@Andrijana:~# docker build -t my-image1 ./
[+] Building 21.7s (7/7) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 86B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/alpine:latest
 => [internal] load build context
 => => transferring context: 62B
 => [1/2] FROM docker.io/library/alpine@sha256:124c7d2707904eea7431fffe91522a01e5a861a624ee31d03372cc1d138a3
 => => resolve docker.io/library/alpine@sha256:124c7d2707904eea7431fffe91522a01e5a861a624ee31d03372cc1d138a3
 => => sha256:124c7d2707904eea7431fffe91522a01e5a861a624ee31d03372cc1d138a3126 1.64kB / 1.64kB
 => => sha256:b6ca290b6b4cdcca5b3db3ffa338ee0285c11744b4a6abaa9627746ee3291d8d 528B / 528B
 => => sha256:9ed4aefc74f6792b5a804d1d146fe4b4a2299147b0f50eaf2b08435d7b38c27e 1.47kB / 1.47kB
 => => sha256:f56be85fc22e46face30e2c3de3f7fe7c15f8fd7c4e5add29d7f64b87abdaa09 3.37MB / 3.37MB
 => => extracting sha256:f56be85fc22e46face30e2c3de3f7fe7c15f8fd7c4e5add29d7f64b87abdaa09
 => [2/2] ADD test.sh /
 => exporting to image
 => => exporting layers
 => => writing image sha256:de68ccce045e5e3851c08b48d8f14f35f004959efbc21683912f8b82d670a4ac
 => => naming to docker.io/library/my-image1
root@Andrijana:~#
```

9. Now spawn a container. Run docker run - -name my-test1 my-image1.

```
root@Andrijana:~# docker run --name my-test1 my-image1
/bin/sh: /bin/bash: not found
```

- I am getting /bin/bash: not found

10. Do a docker ps –a. Do you see your container running? No.

```
root@Andrijana:~# docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS                       PORTS     NAMES
5c909c857230   my-image1     "/bin/sh -c '/bin/ba…"   8 seconds ago    Exited (127) 7 seconds ago             test
7257e3c89857   my-image1     "/bin/sh -c '/bin/ba…"   21 seconds ago   Exited (127) 20 seconds ago            my-test1
10d2ac6810d8   hello-world   "/hello"                 15 minutes ago   Exited (0) 15 minutes ago              festive_mcnulty
3bd6814e12f9   hello-world   "/hello"                 28 minutes ago   Exited (0) 28 minutes ago              zealous_lamport
```

11. Do a docker logs my-test1. What is the output of the log? Note: Because alpine is very light Image it does not have bash binaries.

```
root@Andrijana:~# docker logs my-test1
/bin/sh: /bin/bash: not found
root@Andrijana:~#
```

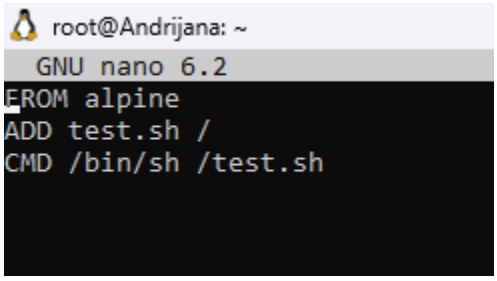
12. Delete my-test. Run docker rm –f my-test1.

```
root@Andrijana:~# rm -f my-test1
root@Andrijana:~#
```

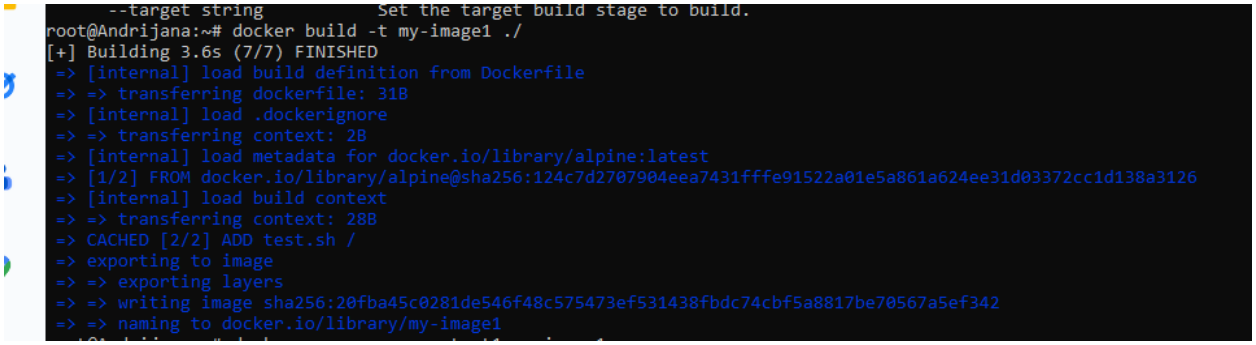
13. Delete my–image. Run docker rmi –f my-image1.

```
root@Andrijana:~# docker rmi -f my-image1
Untagged: my-image1:latest
Deleted: sha256:de68ccce045e5e3851c08b48d8f14f35f004959efbc21683912f8b82d670a4ac
root@Andrijana:~#
```

Now correct your Dockerfile. In the last line replace CMD /bin/bash /test.sh with CMD /bin/sh /test.sh.



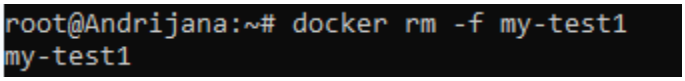
14. Build your image. Run docker build –t my-image1 ./



15. Now spawn a container again. Run docker run - -name my-test1 my-image1.

16. Do a docker ps –a. Do you see your container running? Yes.

17. Delete my-test. Run docker rm –f my-test1.



18. Delete my–image. Run docker rmi –f my-image1.