



UNIVERSITY OF NIŠ
FACULTY OF
ELECTRONIC
ENGINEERING



**Docker in system
administration**

Practical assignment

Student:

Ivan Šušter, br. ind. 1548

Mentor:

Prof. dr Milorad Tošić

Niš, November 2022.

CONTENT

1. INTRODUCTION	1
2. THEORETICAL OVERVIEW AND ASSIGNMENT	1
2.1. Using bind mounts	1
2.1.1. The changes in directory on the host	3
2.1.2. The changes in direcotry from container	4
2.2. Using volumes	5
2.2.1. Manipulating data in container	7
2.2.2. Creating new container with old volume	7
3. CREATING VOLUMES AND CONTAINERS WITH DOCKER COMPOSER.....	8

1. INTRODUCTION

This is practical report for the first practical assignment for course Virtualization. In this report will be practically shown how to use Docker bind mounts and volumes. The full assignment text is below.

Assignment:

On a **host** computer create folder “students”. Create docker image such that any user that is logged in the docker container can access only the folder “students” and no other folder on the **host**.

2. THEORETICAL OVERVIEW AND ASSIGNMENT

Docker offers several ways to manage data between host and container. Two ways offer storing files on the host, so that the files are persisted after container stops. This two ways are **volumes** and **bind mounts**.

Also it is possible to store files in memory on the host, but that files are not persisted. For this is used **tmpfs mount** and **named pipe** depending on which OS is running on host.

Volumes are stored in a part of the host file system which is managed by Docker (`/var/lib/docker/volumes/` on Linux). And this is the best way to persist data in Docker.

Bind mounts may be stored anywhere on the host system. They may even be important system files or directories.¹

2.1. Using bind mounts

First the folder named “students” will be created on the host home folder and in folder it will be created text file with some text in it like on images below.

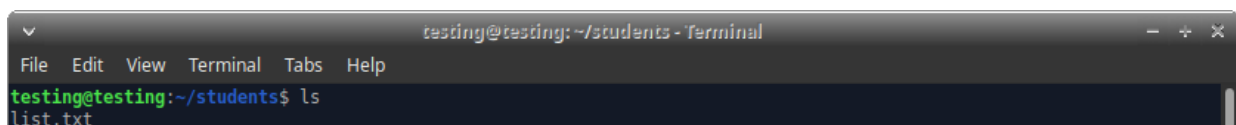
A terminal window titled 'testing@testing: ~/students - Terminal'. The terminal shows the command 'ls' being executed, and the output 'list.txt' is displayed on the next line. The terminal has a menu bar with 'File', 'Edit', 'View', 'Terminal', 'Tabs', and 'Help'.

Figure 1 Files in students directory on the host

¹ <https://docs.docker.com/storage/> (Checked: 20.11.2022.)

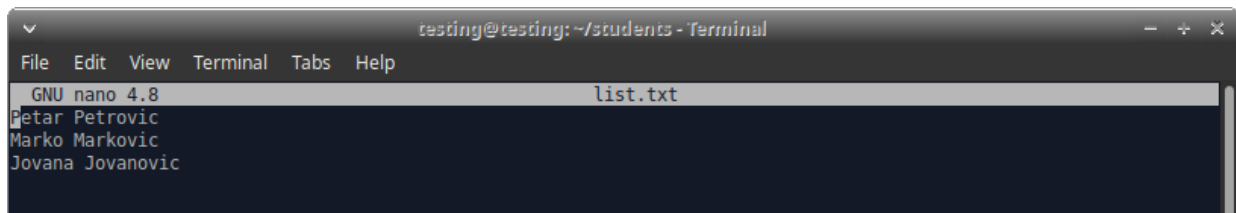


Figure 2 Content of list.txt file on the host

Now Docker container can be started, for this assignment docker alpine linux will be used as image for creating container which is a minimal Docker image based on Alpine Linux only about 5 MB in size.

Container will be started using comand given on next image

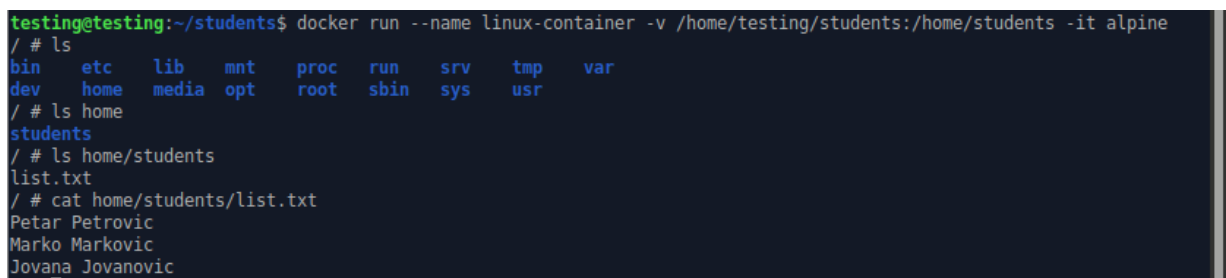


Figure 3 Creating container

docker run – is used for running processes in isolated container, and container is a process which run on host.

Option - **-name** is used for naming a container which can be very usefull to describe what some container represents and so we can refer to it without using container ID. This option is not required if you don't specified it the docker will automaticly give the name of container which will be some random name.

Next option is **-v** which is used for creating bind mount the syntacs for this command is **-v /src/on/host:/dest/on/container**. In this case directory **students** which is locate on the host in **home** direcotry of **testing** user is bind to container's **home** direcotry that way any user in container can access direcotry **students**.

Option **-it** means basicly to use interactive terminal so we can use for example shell to issue commands and read the output from container. Last word in command **alpine** represents the image.

Now that container is started we can issue commands to navigate container's file system which is isloated from host's file system.

As shown on image 3 **ls** command is used for listing the content of **root** and then listing the content from **home** directory and last **ls** command is used to list the content from **students** directory where the file *list.txt* is placed from **host**. Lastly **cat** command was used to show the content of the file *list.txt*.

Running **docker inspect <name_of_container>** command, in this case *name_of_container* is **linux-container**, prints information about that specific container by scrolling down there are information about mounts which is shown on image 4.

```
"Mounts": [
  {
    "Type": "bind",
    "Source": "/home/testing/students",
    "Destination": "/home/students",
    "Mode": "",
    "RW": true,
    "Propagation": "rprivate"
  }
],
```

Figure 4 Container information

Now as shown on image the “**RW**” option is **true** which means that it can be read and written to and from that directory, if read-only is needed for bind mounts, when creating a container in command after destination of the bind mounts **:ro** is added **docker run --name linux-container -v /home/testing/students:/home/students:ro --it alpine**.

2.1.1. The changes in directory on the host

Every change in direcorly **students** on host will reflect to change in **students** directory in container. Now there is only one file in the directory named *list.txt*, by adding new file on the host (*Image 5*) and listing files in the container the new file created on the host will appear in the container which is shown on image. (*Image 6*)

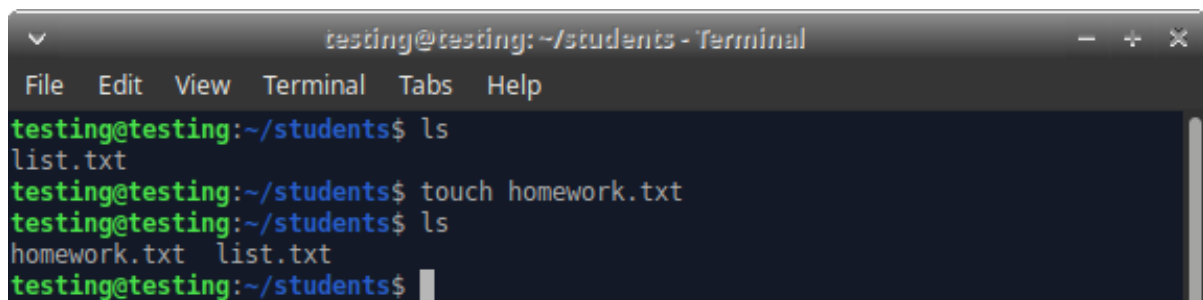
A terminal window titled 'testing@testing: ~/students - Terminal'. The prompt is 'testing@testing:~/students\$'. The user runs 'ls' and sees 'list.txt'. Then they run 'touch homework.txt'. Finally, they run 'ls' again and see 'homework.txt list.txt'.

Figure 5 Creating new file on the host

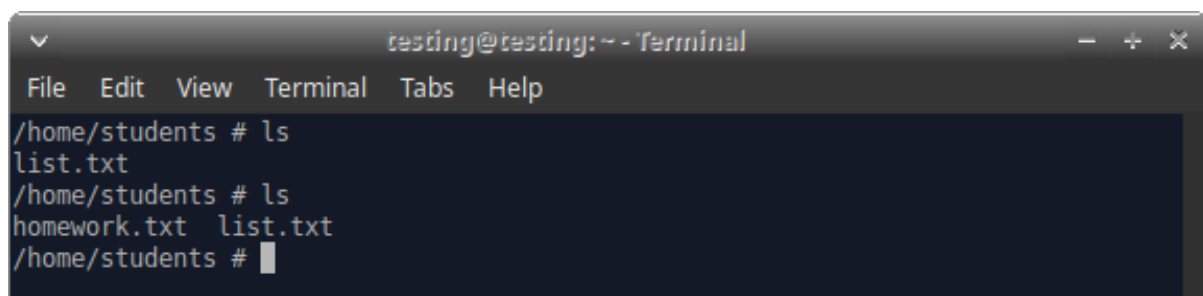
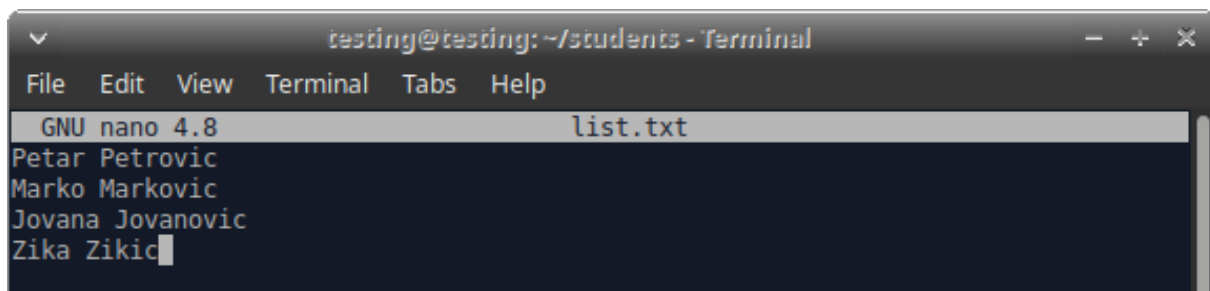
A terminal window titled 'testing@testing: ~ - Terminal'. The prompt is '/home/students #'. The user runs 'ls' and sees 'list.txt'. Then they run 'ls' again and see 'homework.txt list.txt'.

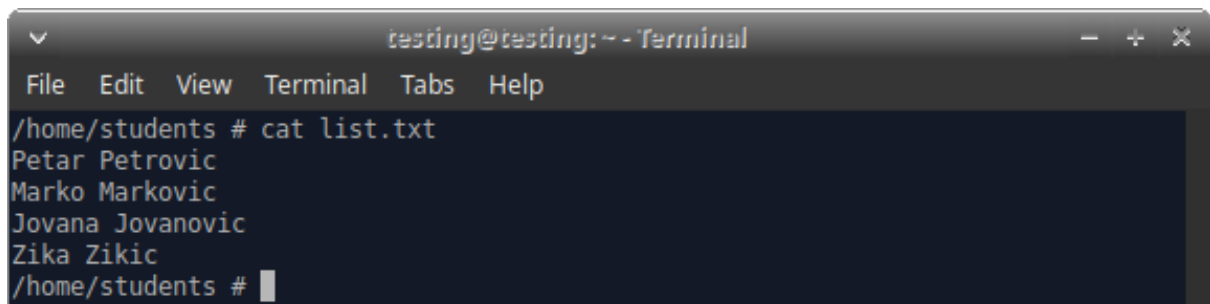
Figure 6 Listing files in container

Also changing content of files for example adding new line in the file *list.txt* on the host (*Image 7*) and saving it will also reflect to container. (*Image 8*)

A terminal window titled 'testing@testing: ~/students - Terminal'. It shows the GNU nano 4.8 editor editing a file named 'list.txt'. The file contains four lines of text: 'Petar Petrovic', 'Marko Markovic', 'Jovana Jovanovic', and 'Zika Zikic'. The cursor is at the end of the last line.

```
testing@testing: ~/students - Terminal
File Edit View Terminal Tabs Help
GNU nano 4.8 list.txt
Petar Petrovic
Marko Markovic
Jovana Jovanovic
Zika Zikic
```

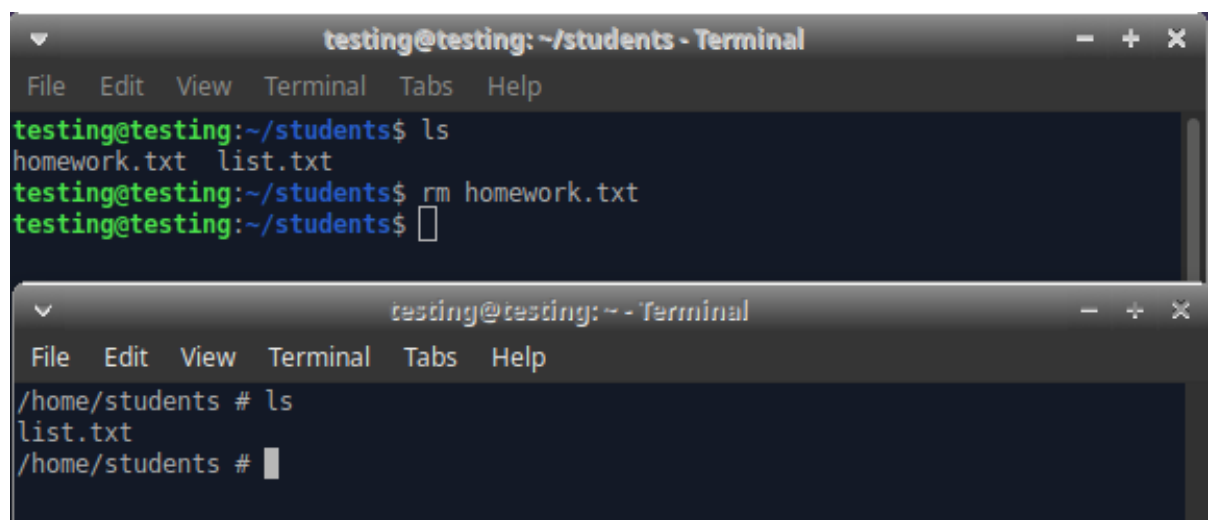
Figure 7 New line in file on the host

A terminal window titled 'testing@testing: ~ - Terminal'. It shows the command 'cat list.txt' being executed in the '/home/students' directory. The output is the same four lines of text as in Figure 7.

```
testing@testing: ~ - Terminal
File Edit View Terminal Tabs Help
/home/students # cat list.txt
Petar Petrovic
Marko Markovic
Jovana Jovanovic
Zika Zikic
/home/students #
```

Figure 8 File in container after change on host

Removing file or content from file works the same way all changes will be reflected to the files in container. (Image 9)

Two terminal windows are shown. The top window is titled 'testing@testing: ~/students - Terminal' and shows the commands 'ls' and 'rm homework.txt' being executed. The output of 'ls' shows 'homework.txt' and 'list.txt'. The output of 'rm' is empty. The bottom window is titled 'testing@testing: ~ - Terminal' and shows the command 'ls' being executed in the '/home/students' directory. The output is 'list.txt'.

```
testing@testing: ~/students - Terminal
File Edit View Terminal Tabs Help
testing@testing:~/students$ ls
homework.txt list.txt
testing@testing:~/students$ rm homework.txt
testing@testing:~/students$

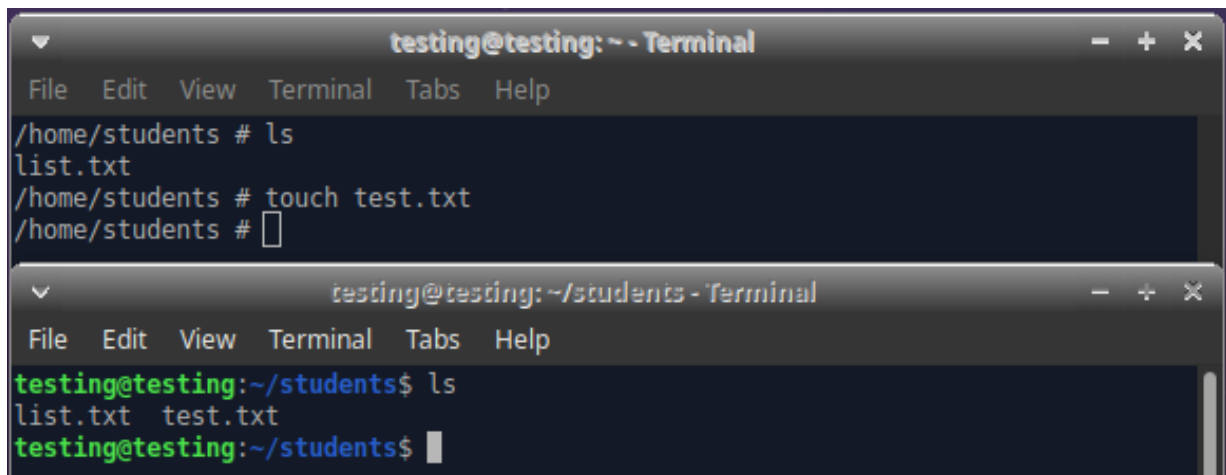
testing@testing: ~ - Terminal
File Edit View Terminal Tabs Help
/home/students # ls
list.txt
/home/students #
```

Figure 9 Removing file on the host

2.1.2. The changes in direcotry from container

All changes that worked from previous chapter (Chapter 2.2.) works both directions which means that changing in the directory **students** from the container will also reflect to host's directory **students**, but only to that specific directory since that's the selected directory binded to the container. Other changes in container will remain only in container as long as the container lives. Which leads to the fact that containers do not persist any data or changes when they are restarted.

On the image below file will be added in the container and then listed in the host direcotry. (Image 10)



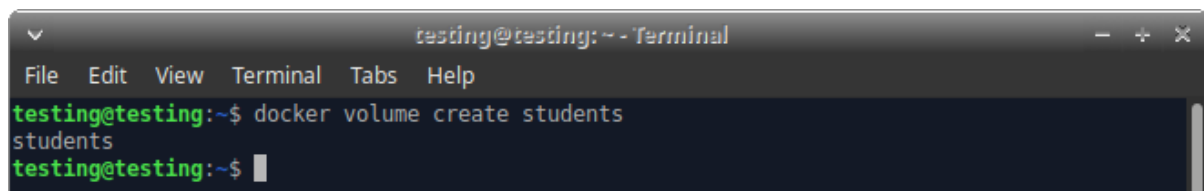
```
testing@testing: ~ - Terminal
File Edit View Terminal Tabs Help
/home/students # ls
list.txt
/home/students # touch test.txt
/home/students #

testing@testing: ~/students - Terminal
File Edit View Terminal Tabs Help
testing@testing:~/students$ ls
list.txt test.txt
testing@testing:~/students$
```

Figure 10 Adding file in container

2.2. Using volumes

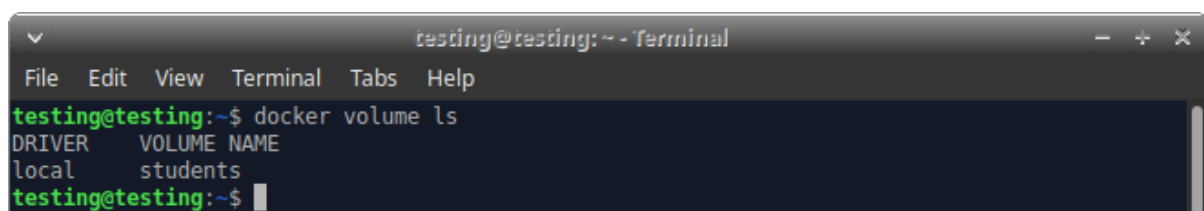
Using a volumes is preferably way to persist data generated by and used by containers. Using volumes is similar to using bind mounts. First command is **docker volume create** <name_of_volume> in this case name of the volume is **students**. (Image 11)



```
testing@testing: ~ - Terminal
File Edit View Terminal Tabs Help
testing@testing:~$ docker volume create students
students
testing@testing:~$
```

Figure 11 Creating volume

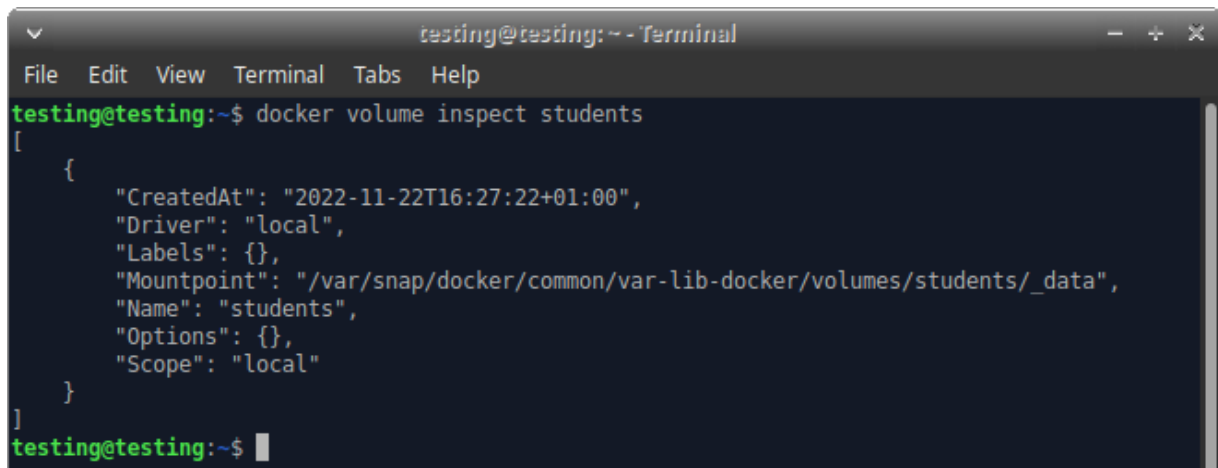
To check if the volume is created command **docker volume ls** can be issued. (Image 12)



```
testing@testing: ~ - Terminal
File Edit View Terminal Tabs Help
testing@testing:~$ docker volume ls
DRIVER      VOLUME NAME
local       students
testing@testing:~$
```

Figure 12 Listing existing volumes

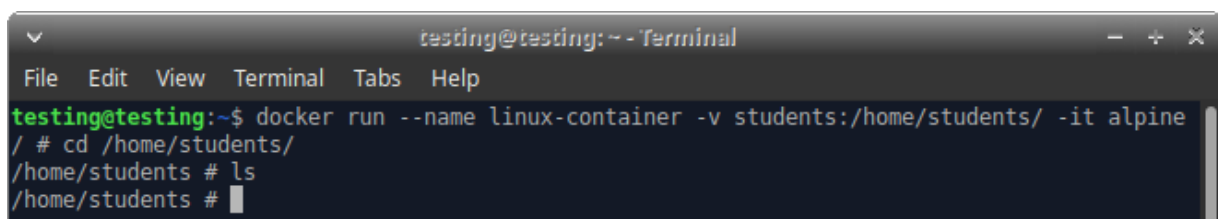
Also command **docker volume inspect** <name_of_volume> can be issued to display information about volume. Here the **mountpoint** represent absolute path to created volume which is in `/var/snap/docker/common/var-lib-docker/volumes/students/_data`. (Image 13)

A terminal window titled 'testing@testing: ~ - Terminal' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The command 'docker volume inspect students' has been executed, resulting in a JSON output showing details for the 'students' volume, including its creation time, driver, labels, mountpoint, name, options, and scope.

```
testing@testing:~$ docker volume inspect students
[
  {
    "CreatedAt": "2022-11-22T16:27:22+01:00",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/snap/docker/common/var-lib-docker/volumes/students/_data",
    "Name": "students",
    "Options": {},
    "Scope": "local"
  }
]
testing@testing:~$
```

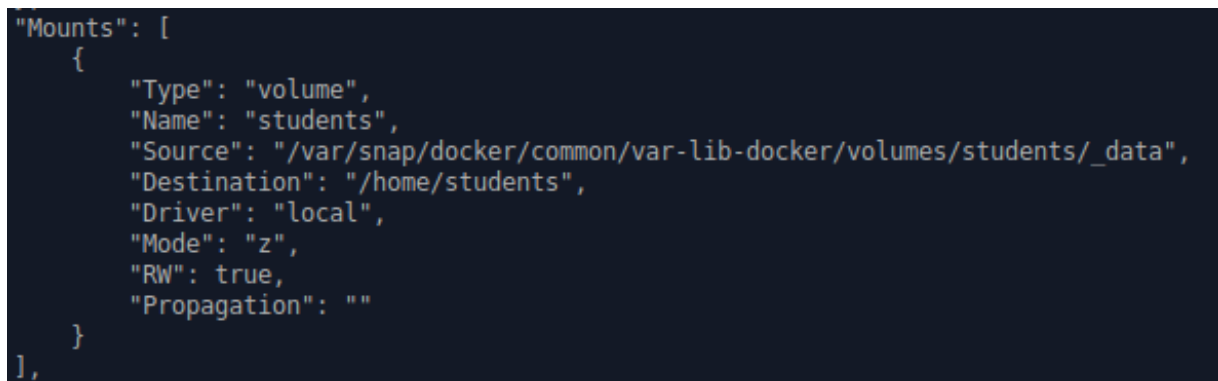
Figure 13 Details of volume students

Now that volume is created it is time to start container, again the same image will be used and command is **docker run --name linux-container -v students:/home/students/ -it alpine**.(Image 14) First, container will be inspected to see if there is volume attached to container. (Image 15)

A terminal window titled 'testing@testing: ~ - Terminal' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The command 'docker run --name linux-container -v students:/home/students/ -it alpine' has been executed. The prompt changes to '/ #', and the user enters 'cd /home/students/' and 'ls', showing the contents of the mounted volume.

```
testing@testing:~$ docker run --name linux-container -v students:/home/students/ -it alpine
/ # cd /home/students/
/home/students # ls
/home/students #
```

Figure 14 Creating and running container

A terminal window showing the output of 'docker inspect linux-container'. The output is a JSON array containing details about the container, including its name, image, command, and a 'Mounts' section that shows the volume 'students' is mounted to '/home/students' with read-write permissions.

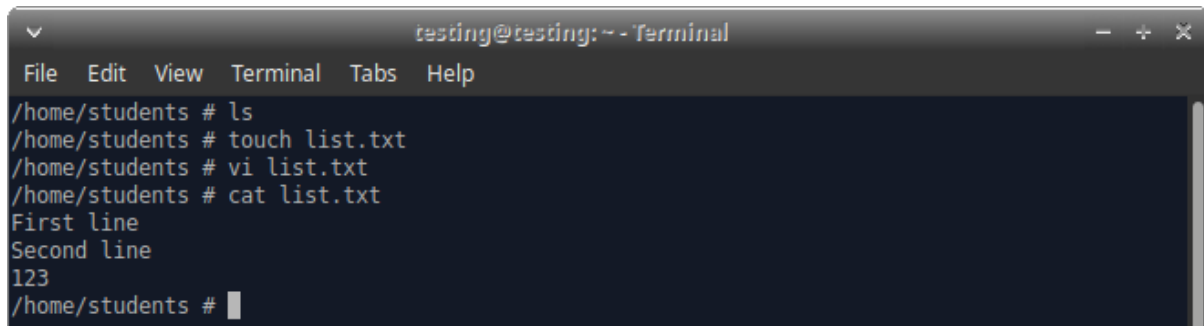
```
"Mounts": [
  {
    "Type": "volume",
    "Name": "students",
    "Source": "/var/snap/docker/common/var-lib-docker/volumes/students/_data",
    "Destination": "/home/students",
    "Driver": "local",
    "Mode": "z",
    "RW": true,
    "Propagation": ""
  }
],
```

Figure 15 Details of running container

Now as shown on image the “RW” option is **true** which means that it can be read and written to that volume, if read-only is needed for volume, when creating a container in command after destination of the volume in container **:ro** is added **docker run -v students:/home/students/:ro --name linux-container -it alpine**.

2.2.1. Manipulating data in container

Creating files in container in the directory which is bind to volume is shown on image below. That is the way to persist data from the container. Simple example of creating text file is given on image below. (Image 16)

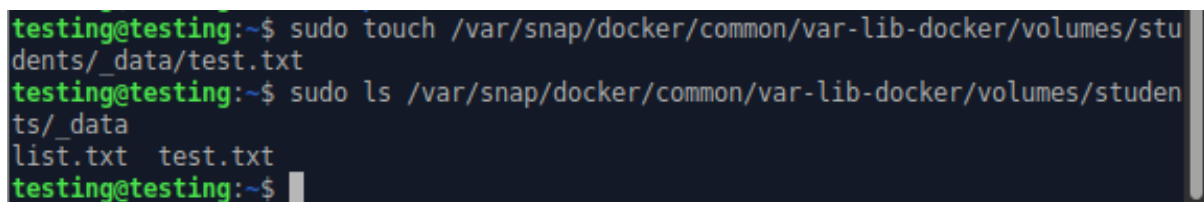
A terminal window titled 'testing@testing: ~ - Terminal' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the following commands and output:

```
/home/students # ls
/home/students # touch list.txt
/home/students # vi list.txt
/home/students # cat list.txt
First line
Second line
123
/home/students #
```

Figure 16 Creating and writing in file

On the image above is listed content which is persisted on host.

It is possible to create file on the host volume but **root permission** is required. (Image 17)

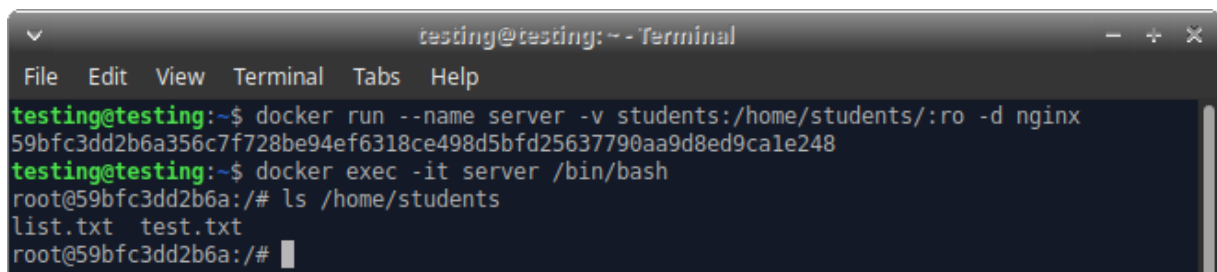
A terminal window showing the following commands and output:

```
testing@testing:~$ sudo touch /var/snap/docker/common/var-lib-docker/volumes/stu
dents/_data/test.txt
testing@testing:~$ sudo ls /var/snap/docker/common/var-lib-docker/volumes/studen
ts/_data
list.txt  test.txt
testing@testing:~$
```

Figure 17 Creating file on host in docker volume

2.2.2. Creating new container with old volume

Now that data is persist from container created in the previous chapter new container can be created with same volume, this way is used to share data between containers. This time container will be created by using **nginx** image by issuing following command **docker run --name server -v students:/home/students/:ro -d nginx**. (Image 18)

A terminal window showing the following commands and output:

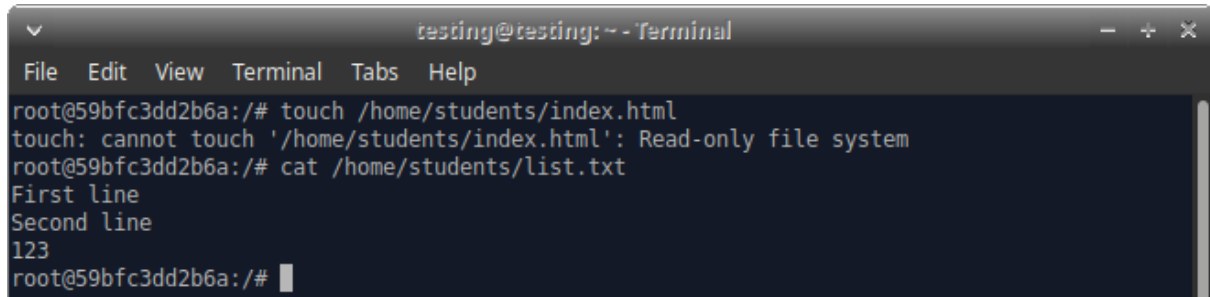
```
testing@testing:~$ docker run --name server -v students:/home/students/:ro -d nginx
59bfc3dd2b6a356c7f728be94ef6318ce498d5bfd25637790aa9d8ed9ca1e248
testing@testing:~$ docker exec -it server /bin/bash
root@59bfc3dd2b6a:/# ls /home/students
list.txt  test.txt
root@59bfc3dd2b6a:/#
```

Figure 18 Creating new container

In this command after destination of volume in container **:ro** is added which means that in container data from volume are read-only. Also new option **-d** means to run container in

detach mode which means to run the container in the background this option print output and that output is the ID of the container. Following command is **docker exec -it server /bin/bash** which means to run command in running container using **bash** shell.

Proof that read-only is possible from this container is given on image below. (Image 19)

A terminal window titled 'testing@testing: ~ - Terminal' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows a user at a container root@59bfc3dd2b6a attempting to create a file and view its contents. The 'touch' command fails with a 'Read-only file system' error, while the 'cat' command successfully displays the contents of a file.

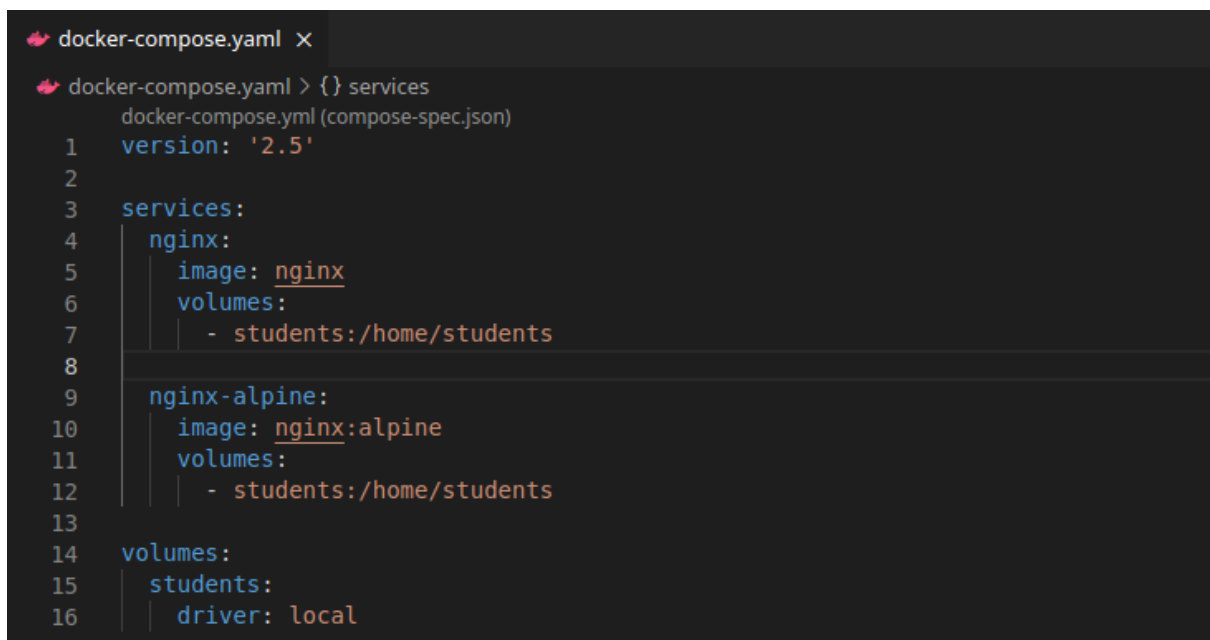
```
root@59bfc3dd2b6a:/# touch /home/students/index.html
touch: cannot touch '/home/students/index.html': Read-only file system
root@59bfc3dd2b6a:/# cat /home/students/list.txt
First line
Second line
123
root@59bfc3dd2b6a:/#
```

Figure 19 Read-only volume

3. CREATING VOLUMES AND CONTAINERS WITH DOCKER COMPOSER

The way of creating volumes described in the previous chapter is good for testing but in production the easiest way to create containers and volumes is by using Docker compose especially when we need more than one container. In YAML file services can be defined can all that services can be created and destroyed by one command.

The next image represent example of YAML file for creating 2 containers which can share same volume from host.

A screenshot of a code editor showing a Docker Compose YAML file named 'docker-compose.yml'. The file defines two services, 'nginx' and 'nginx-alpine', both sharing a volume named 'students'. The 'nginx' service uses the 'nginx' image, and 'nginx-alpine' uses the 'nginx:alpine' image. Both are connected to the 'students' volume, which is configured with the 'local' driver.

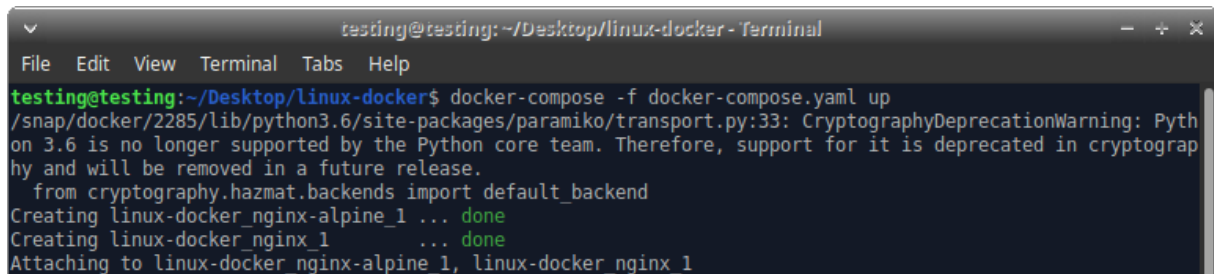
```
docker-compose.yml > {} services
docker-compose.yml (compose-spec.json)
1  version: '2.5'
2
3  services:
4    nginx:
5      image: nginx
6      volumes:
7        - students:/home/students
8
9    nginx-alpine:
10     image: nginx:alpine
11     volumes:
12       - students:/home/students
13
14  volumes:
15    students:
16     driver: local
```

Figure 20 Docker compose YAML file

Two containers will be created with different version of nginx the first one will be latest

version, and the second will be alpine version of nginx. Both will share same volume named students.

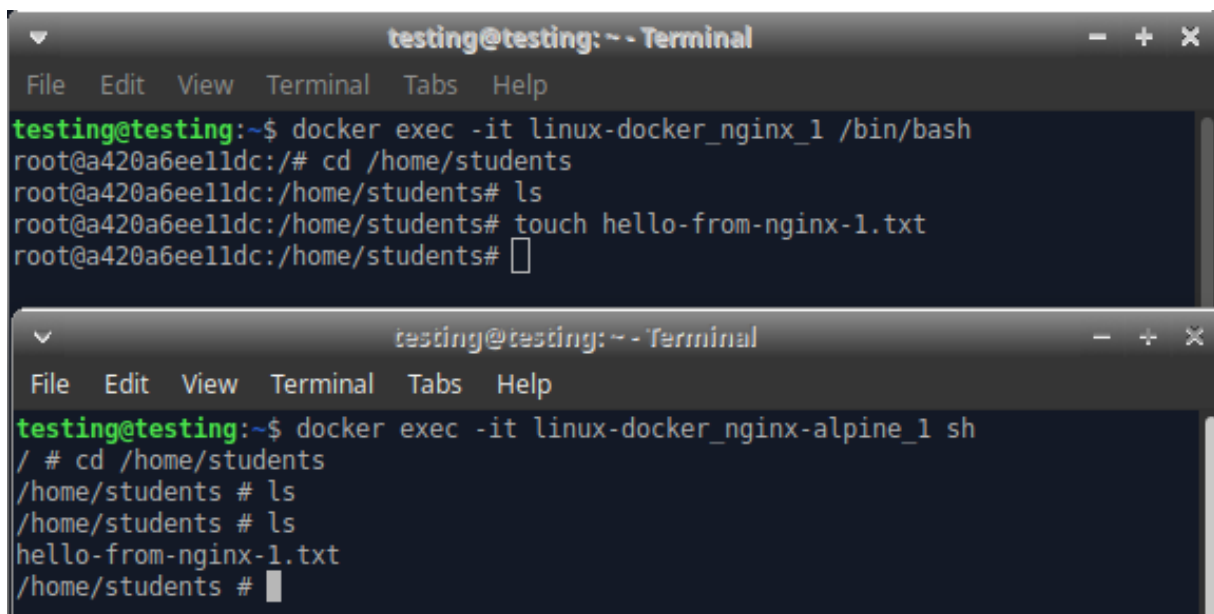
Image 21 shows how to run docker compose file in other words it will start all services defined in docker compose file.



```
testing@testing: ~/Desktop/linux-docker - Terminal
File Edit View Terminal Tabs Help
testing@testing:~/Desktop/linux-docker$ docker-compose -f docker-compose.yaml up
/snap/docker/2285/lib/python3.6/site-packages/paramiko/transport.py:33: CryptographyDeprecationWarning: Pyth
on 3.6 is no longer supported by the Python core team. Therefore, support for it is deprecated in cryptograp
hy and will be removed in a future release.
  from cryptography.hazmat.backends import default_backend
Creating linux-docker_nginx-alpine_1 ... done
Creating linux-docker_nginx_1 ... done
Attaching to linux-docker_nginx-alpine_1, linux-docker_nginx_1
```

Figure 21 Docker compose command

After containers are up and running command **docker exec** can be issued to access container. Both containers are shown on image below as well creating file on the first container and listing it on the second container.

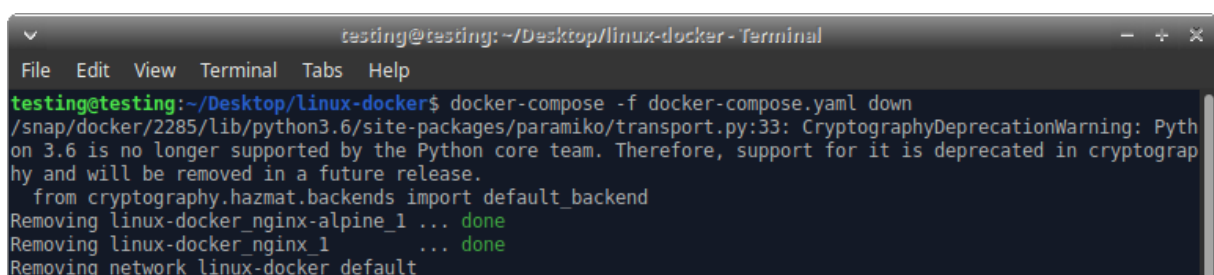


```
testing@testing: ~ - Terminal
File Edit View Terminal Tabs Help
testing@testing:~$ docker exec -it linux-docker_nginx_1 /bin/bash
root@a420a6ee11dc:/# cd /home/students
root@a420a6ee11dc:/home/students# ls
root@a420a6ee11dc:/home/students# touch hello-from-nginx-1.txt
root@a420a6ee11dc:/home/students#

testing@testing: ~ - Terminal
File Edit View Terminal Tabs Help
testing@testing:~$ docker exec -it linux-docker_nginx-alpine_1 sh
/ # cd /home/students
/home/students # ls
/home/students # ls
hello-from-nginx-1.txt
/home/students #
```

Figure 22 Running containers

At the end when containers are no longer needed command shown on image below can be issued for deleting them.



```
testing@testing: ~/Desktop/linux-docker - Terminal
File Edit View Terminal Tabs Help
testing@testing:~/Desktop/linux-docker$ docker-compose -f docker-compose.yaml down
/snap/docker/2285/lib/python3.6/site-packages/paramiko/transport.py:33: CryptographyDeprecationWarning: Pyth
on 3.6 is no longer supported by the Python core team. Therefore, support for it is deprecated in cryptograp
hy and will be removed in a future release.
  from cryptography.hazmat.backends import default_backend
Removing linux-docker_nginx-alpine_1 ... done
Removing linux-docker_nginx_1 ... done
Removing network linux-docker_default
```

Figure 23 Removing containers with docker compose