htl donaustadt
Donaustadtstraße 45
1220 Wien

Abteilung: Informationstechnologie
Schwerpunkt: Netzwerktechnik

Thema

Laboratory protocol GNU/Linux - Setting up a multi-user environment

Figure 1: Grouplogo

Subject: ITSI|ZIVK
Class: 3AHITN
Name: Stefan Fürst, Marcel Raichle
Groupname/number: Dumm und Dümmer/7
Supervisor: ZIVK
Exercise dates:
Submission date:

htl donaustadt
Donaustadtstraße 45
1220 Wien

Abteilung: Informationstechnologie
Schwerpunkt: Netzwerktechnik

Contents

1 Task definition

2 Summary

3 Exercise execution

3.1 creating the container

I choose to write my own Dockerfile for this, which is a textfile, that describes the commands needed to create the desired image. Lets walkthrough how to create an image for the first task.
We start by using the `FROM` keyword to specify from the base image [4] from which we start.
`FROM ubuntu:latest`
I choose the ubuntu image [5] and used the `latest` tag, which points to the latest LTS release.
However if we were to build, start and exec into the container we wouldnt be able to do it beacouse it would imidiatly showdown, since nothing is running.

To midigate this we add `CMD tail -F /dev/null` at the end of out Dockerfile. The `tail` command outputs the last 10 lines of a file and the `-F` argument, which stands for follow so it runs forever outputting the last 10 lines of a given file.[3] I used file `/dev/null` which is a virtual device, that any data that gets written to vanishes. [8] So we essentially read an empty file forever to keep the container up.

If we now run the following commands to build the image, run the container and get a shell in it.

```
#build the image
docker buildx build -t image-name .
#run the container
docker run -d --name container-name
#exec into the container (get a shell in it)
docker exec -it container-name /bin/bash
```

To make to commands less work to type, i like to make a shell script that i can source to have aliases for it like this.

```
#!/bin/sh
alias relaunch="sudo sh -c 'docker stop itsi &&\
      docker rm itsi &&\
      docker buildx build -t itsi:latest . &&\
      docker run -d -p 38452:38452 --name itsi itsi:latest &&\
      docker exec -it itsi /bin/bash'"
alias rebuild="sudo sh -c 'docker buildx build -t itsi:latest . &&\
      docker run -d -p 38452:38452 --name itsi itsi:latest &&\
      docker exec -it itsi /bin/bash'"
alias stop="sudo sh -c 'docker stop itsi && docker rm itsi'"
```

Now we are in the container but it has none of the required Packages install that are required for this exercise. They can be installed now in the container which would defeat the entire purpose of making an image, so we use the `RUN` keyword in our Dockerfile along with the desired command to run it when the image is build so the packages are installed as soon as you spin up the container:

```
RUN apt update
RUN apt upgrade -y
RUN apt install iproute2 iputils-ping zsh net-tools vim -y
```

3.2 testing connectivity

Now we can finally test the connectivity, since have the `iputils-ping` package installed. Everything works out of the box by using the default bridge [1]

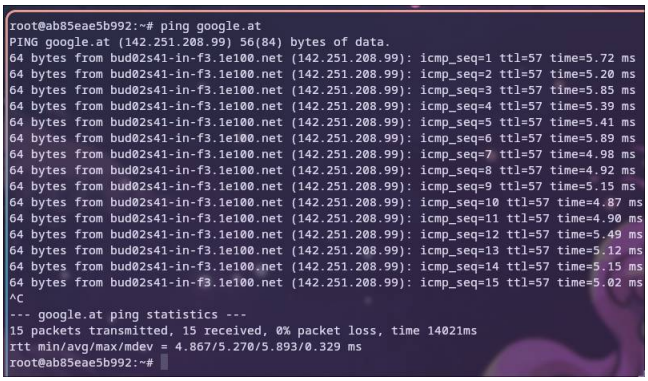

Figure 2: ping to the internet

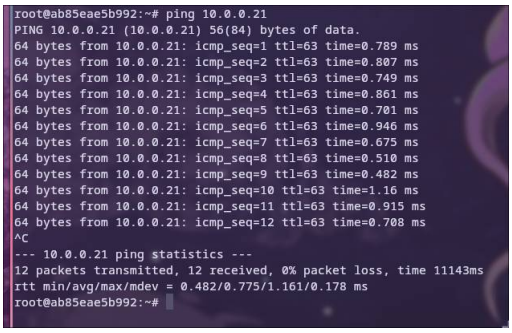

Figure 3: ping to the localnetwork

3.2.1 it works but why?

If we inspect our container by using `docker inspect container-name`, we see that its ip is different than the ones from the lan.

Figure 4: docker inspect

This happens because when you install docker, a virtual interface `docker0` is created, which is used as a networkbridge to allow the container to communicate to the internet and lan. [9]

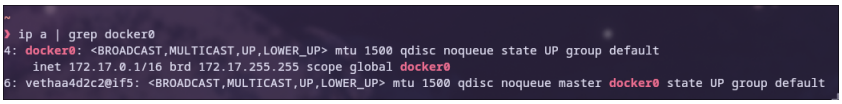

Figure 5: ip a | grep docker0

There are other types of docker networks, but they arent relevant for this exercise.[9]

3.3 creating and managing users

For creating grous/users there are the commands `groupadd` and `useradd`. To add the user we add the following lines to our Dockerfile

```
#creating the group
RUN groupadd -g 324 ram-Users
#creating the users -u is used to set the groupid
RUN useradd -u 1024 ram-alois &&\
    useradd -u 1124 ram-berta &&\
    useradd -u 1224 ram-chris &&\
    useradd ram-fus &&\
    useradd ram-ram
#adding the users to the groups
RUN usermod -g ram-Users ram-alois &&\
    usermod -g ram-Users ram-berta
#settings chris's default shell to zsh
RUN usermod --shell /bin/zsh ram-chris
```

3.3.1 logging in as the users

To login as a different user the `su` command is used.

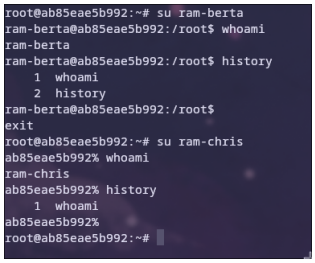

Figure 6: logging in as berta and chris

Each user has their own history, which is saved in their home directory in either the `.bash_history` or `.zsh_history` file. You end the session by using the `exit` command or pressing `<C-d>`.

3.4 setting permissions for directories

The directorys are created with this commad and the `-p` stands for parents and creates parent directorys if needed. For example `mkdir /test/test2` wouldn't work if you dont have `/test` but if you use `mkdir -p` instead it creates `/test` and `/test/test2`.

```
RUN mkdir -p /data/fus &&\
    mkdir /data/fus/alois &&\
    mkdir /data/fus/berta &&\
    mkdir /data/fus/chris &&\
    mkdir /data/fus/public
```

To set the permission three tools will be used: `chrgrp`, `chown` and `chmod`.
First we want everyone in the group to have acess to the directory for this `chgrp -R ram-Users /data/fus/` is used with the `-R` argument meaning recursive [14, 2]
To give everyone every permission in their own directory, we need to make them the owner of it by using `chown -R username:groupname /data/fus/name-of-their-directory`.[12]
Now we can assign the permissions to each directory, for this the `chmod`[10] command is used.
To better understand the command here is a breakdown of the options:

```
u = user who owns the file
g = group -> everyone in the group of the owner
o = other -> everyone else
r = read
w = write
x = execute
+ adding permissions
- removing permissions
= setting permissions
```

[10] Now lets use this to set the permissions accordingly

```
#giving the [g]roup [r]ead and [w]rite permissions for /data/fus
chmod g+rw /data/fus/
#giving the owner all permissions, the [g]roup only read [r]ead and none to [o]thers
chmod -R u+wrx,g=r,o= /data/fus/alois/
#same for berta
#giving the owner all permissions and none to the [g]roup and [o]thers
chmod -R u+wrx,g=,o= /data/fus/chris/
#giving the owner and [g]roup all permissions and none to [o]thers
chmod -R u+wrx,g+wrx,o=r /data/fus/public/
```

[10]

Figure 7: testing the permissions

If we login as the users we can see that everything works as intended.

3.5 setting up ssh

The two new users required for this have already been created above in 3.3
To setup an ssh server we need to install the package if we just add `ssh` to our install command in the Dockefile we find out, that this commands requires interactions to set the timezone we need to add these two extra lines to the Dockerfile.

```
#setting the timezone
RUN ln -fs /usr/share/zoneinfo/Europe/Vienna /etc/localtime
#running the command without it beeing interactive
RUN DEBIAN_FRONTEND=noninteractive apt install -y tzdata ssh
```

Now ssh is installed, but it needs to be started for this all we need to do is edit the last line of the file to launch the service aswell.

```
#the default command from before
CMD tail -F /dev/null
#with starting ssh
CMD service ssh start && tail -F /dev/null
```

To find out on what port the server is listening for ssh we use the netstat command, which comes with the `net-tools` package, which we installed earlier. For this the command `netstat -tunlp | grep ssh` is used. Here are the options of the command explained.

```
-t show TCP ports
-u show UDP ports
-n show numerical addresses instead of resolving hosts
-l show only listening ports
-p show the PID of the listener's process
```

[11]

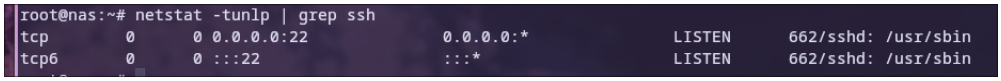

Figure 8: finding the port with netstat

Appereantly its a ”good practise” to move off the standart ssh port and use a different port to evade bots and script kiddies who are scanning the internet for public servers with ssh and testing default passwords. I belive that this is snake oil to change to port for better security, because if you disable password authentication, have a strong password or ban failing ips with tools such as fail2ban all of the problems are resolved anyway.[13]
I still changed to port to show how it would be done anyway. For this we need to edit the file `/etc/ssh/sshd_config`. To do this we can use the preinstaleld texteditor `sed` so edit the file with the following command so we can change the port in the Dockerfile

```
#-i edit the file in place without printing it to the console
#s to use the substitute command of sed
#'/s/string-you-want-to-replace/string-you-want-to-replace-it-with'
#/etc/ssh/sshd_config file that you want to edit
RUN sed -i 's/#Port 22/Port 38452/' /etc/ssh/sshd_config
```

If we try to ssh in with the created user we cant yet, since we havent published any ports in our container yet.

3.5.1 logging into the ssh server

To do this we need to add one line to the Dockerfile and edit the docker run command.

```
#add this to the with the port you choose to the Dockerfile
EXPOSE 38452
#add -p to [p]ublish the desired port
docker run -d -p 38452 --name container -name
```

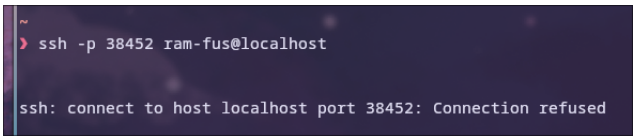

Figure 9: connection refused

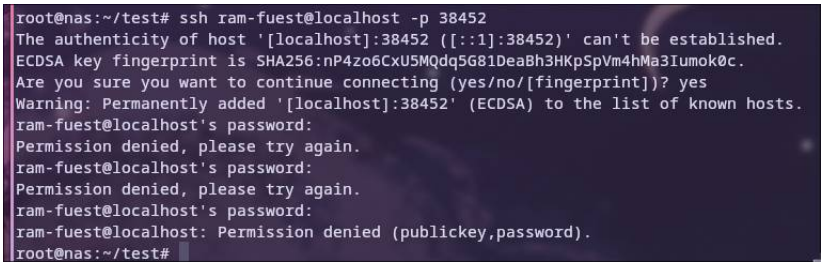

Figure 10: logging without a password

Even if we login now it will still not work, since the user doens't have a password.
To fix this we add this line to our Dockerfile:

```
RUN echo 'root:youresecurepasswordhere' | chpasswd
```

We change the root password instead of the user password, since we dont have `sudo` setup and having to type sudo for every command, when we are the only user is both unecessary and anoing.

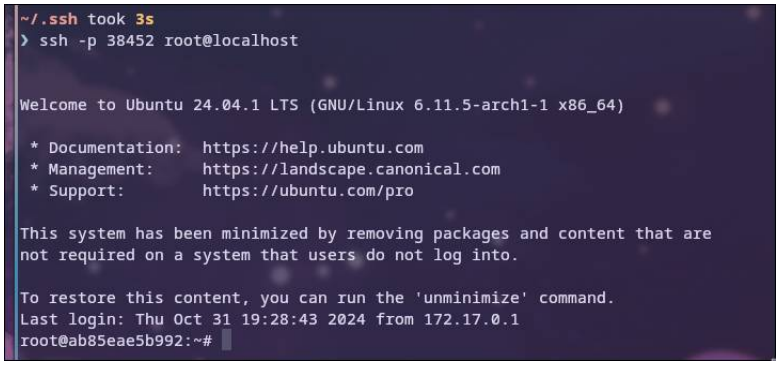

Figure 11: working login

3.5.2 enabeling keypair authenthication

To generate a keypair, we go back to out host system and run the comand `ssh-keygen -b 4096` to generate an 4096-bit SSH-Key. [7] On linux the keys get saved to the `/.ssh` directory, but you can specify a location with `-f`. The file ending with `.pub` is the public key and the other one is the private key.

Figure 12: keys in the directory

To copy the public key to the server, which we want to use it we the `ssh-copy-id` command on linux and `scp` on windows and mac. [6]

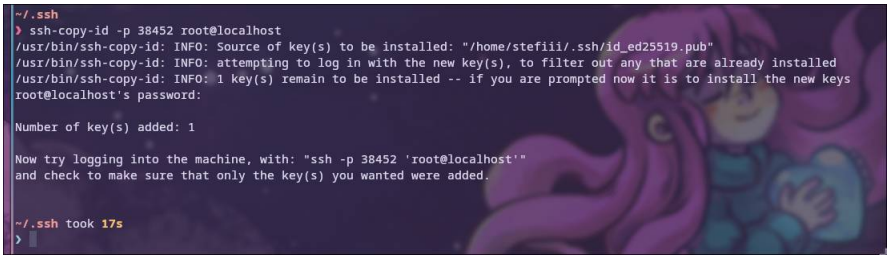

Figure 13: ssh-copy-id

After this if we dont need to enter a password anymore to authenticate.

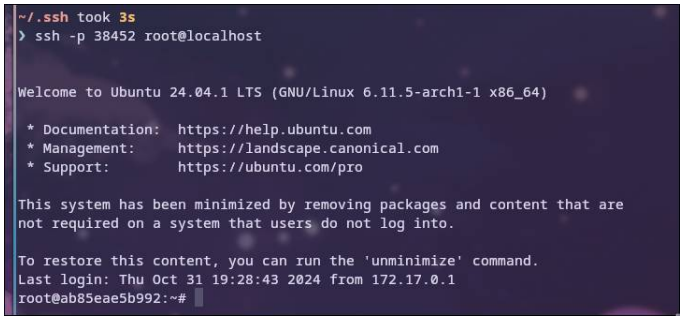

Figure 14: logging with a key

3.5.3 disableling password authentication

To only allow authetication via keys, we need to edit the file `/etc/ssh/sshd_config` again. To do this we ssh into the server and use open the file with the texteditor of your choice and edit this line.

```
#change this
#PasswordAuthentication yes
#to this
PasswordAuthentication no
```

If we try to login as another user, for which we dont have a key we cant connect.

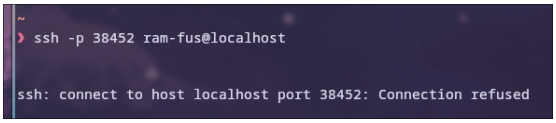

Figure 15: not having a key

References

[1] ”Bridge network driver”, September 2024. [Online; accessed 1. Nov. 2024].

[2] cheat.sh/chgrp, November 2024. [Online; accessed 3. Nov. 2024].

[3] cheat.sh/tail, November 2024. [Online; accessed 1. Nov. 2024].

[4] ”Glossary”, August 2024. [Online; accessed 1. Nov. 2024].

[5] ubuntu - Official Image, November 2024. [Online; accessed 1. Nov. 2024].

[6] Admin. What is ssh-copy-id? How ssh-copy-id works? *SSH Communications Security*, February 2023.

[7] Admin. What is ssh-keygen & How to Use It to Generate a New SSH Key? *SSH Communications Security*, February 2023.

[8] GeeksforGeeks. What is /Dev/Null in Linux? *GeeksforGeeks*, July 2023.

[9] Christian Lempa. Docker Networking Tutorial, ALL Network Types explained!, October 2021. [Online; accessed 1. Nov. 2024].

[10] Linuxize. Chmod Command in Linux (File Permissions). *Linuxize*, September 2019.

[11] Linuxize. How to Check for Listening Ports in Linux (Ports in use). *Linuxize*, June 2020.

[12] Linuxize. Chown Command in Linux (File Ownership). *Linuxize*, December 2023.

[13] LiveOverflow. How To Protect Your Linux Server From Hackers!, April 2021. [Online; accessed 3. Nov. 2024].

[14] J. T. McGinty. A Complete Guide to Linux File Ownership and Groups. *MUO*, March 2022.

4 List of figures

List of Figures