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: 25.10.2024, 1.11.2024, 3.11.2024, 6.11.2024
Submission date: 6.11.2024

Contents

1 Task definition **3**

2 Summary **3**

3 Exercise Execution **4**

3.1 Creating the Container . . . 4

3.2 Testing Connectivity . . . 5

3.2.1 It works, but why? . . . 6

3.3 creating and managing users . . . 7

3.3.1 logging in as the users . . . 7

3.4 Set directory privileges . . . 7

3.5 setting up ssh . . . 9

3.5.1 Logging On to the SSH Server . . . 9

3.5.2 enabeling keypair authenthication . . . 11

3.5.3 Disable password authentication . . . 12

4 List of figures **14**

5 Attachments **15**

1 Task definition

Setting up a headless Linux installation with multiple users, adding them to a group, and setting permissions over a directory structure. You will also need to set up an ssh server for which you will need to set up key pair authentication.

2 Summary

To accomplish this, we set up a Docker image that does all the necessary setup so that the container can be rebuilt at any time for easier testing instead of using a heavier vm. To make it easier to rebuild and restart the container, we wrote a shell script to the source so that we had aliases for all the commands. We used the Ubuntu Docker image as a base, installed the required packages since the image comes with a minimal amount of packages, and used `useradd, usermod, chmod, chown, chgrp, su` to add users, change file ownership, permissions and test. Finally, for the ssh part, the service was set up and configured appropriately. We did everything that made sense in the Dockerfile file to make it reproducible.

3 Exercise Execution

3.1 Creating the Container

I decided to write my own dockerfile for this, which is a text file that describes the commands needed to create the desired image. Lets walk through how to create an image for the first task.
We start by using the `FROM` keyword to specify the base image [4] from which we are starting.
`FROM ubuntu:latest`
I chose the ubuntu image [5] and used the `latest` tag, which points to the latest LTS release.
However, if we were to build, start, and execute in the container, we would not be able to do it because it would immediately showdown since nothing is running.
To mitigate this, we add `CMD tail -F /dev/null` to the end of our Dockerfile. The `tail` command prints the last 10 lines of a file and the `-F` argument stands for follow, so it will run forever and print the last 10 lines of a given file.[3] I used the file `/dev/null`, which is a virtual device, so any data written to it will disappear. [6] So we are essentially reading 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.

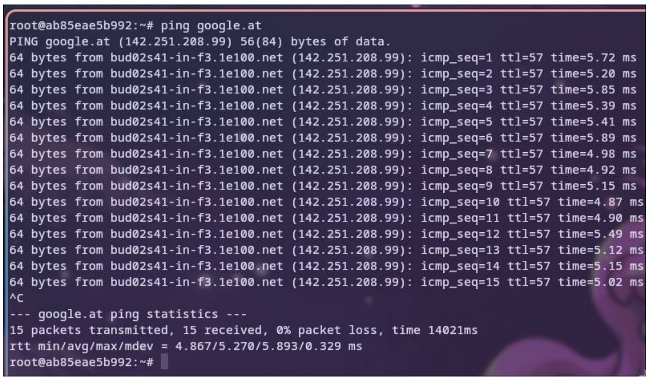
```
#!/bin/sh
alias relaunch="sudo sh -c 'docker stop itsi &&\
      docker rm itsi &&\
      docker buildx build -t itsi:latest . &&\
      docker run -d -p 38452:38452 --name itsi itsi:latest &&\
      docker exec -it itsi /bin/bash'"
alias rebuild="sudo sh -c 'docker buildx build -t itsi:latest . &&\
      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 does not have any of the required packages installed that are needed for this exercise. They can be installed in the container now, which would defeat the whole purpose of building an image, so we use the `RUN` keyword in our dockerfile along with the desired command to run it when the image is built, so that 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
```

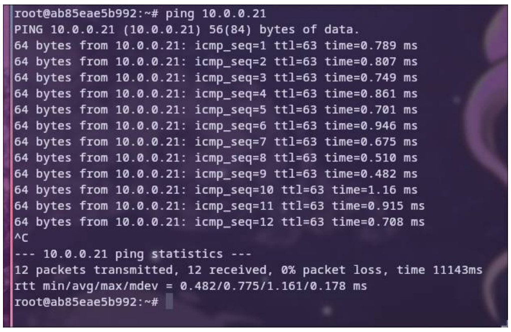
3.2 Testing Connectivity

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



```
root@ab85eae5b992:~# ping google.at
PING google.at (142.251.208.99) 56(84) bytes of data.
64 bytes from bud02s41-in-f3.1e100.net (142.251.208.99): icmp_seq=1 ttl=57 time=5.72 ms
64 bytes from bud02s41-in-f3.1e100.net (142.251.208.99): icmp_seq=2 ttl=57 time=5.20 ms
64 bytes from bud02s41-in-f3.1e100.net (142.251.208.99): icmp_seq=3 ttl=57 time=5.85 ms
64 bytes from bud02s41-in-f3.1e100.net (142.251.208.99): icmp_seq=4 ttl=57 time=5.39 ms
64 bytes from bud02s41-in-f3.1e100.net (142.251.208.99): icmp_seq=5 ttl=57 time=5.41 ms
64 bytes from bud02s41-in-f3.1e100.net (142.251.208.99): icmp_seq=6 ttl=57 time=5.89 ms
64 bytes from bud02s41-in-f3.1e100.net (142.251.208.99): icmp_seq=7 ttl=57 time=4.98 ms
64 bytes from bud02s41-in-f3.1e100.net (142.251.208.99): icmp_seq=8 ttl=57 time=4.92 ms
64 bytes from bud02s41-in-f3.1e100.net (142.251.208.99): icmp_seq=9 ttl=57 time=5.15 ms
64 bytes from bud02s41-in-f3.1e100.net (142.251.208.99): icmp_seq=10 ttl=57 time=4.87 ms
64 bytes from bud02s41-in-f3.1e100.net (142.251.208.99): icmp_seq=11 ttl=57 time=4.90 ms
64 bytes from bud02s41-in-f3.1e100.net (142.251.208.99): icmp_seq=12 ttl=57 time=5.49 ms
64 bytes from bud02s41-in-f3.1e100.net (142.251.208.99): icmp_seq=13 ttl=57 time=5.12 ms
64 bytes from bud02s41-in-f3.1e100.net (142.251.208.99): icmp_seq=14 ttl=57 time=5.15 ms
64 bytes from bud02s41-in-f3.1e100.net (142.251.208.99): icmp_seq=15 ttl=57 time=5.02 ms
^C
--- google.at ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14021ms
rtt min/avg/max/mdev = 4.867/5.270/5.893/0.329 ms
root@ab85eae5b992:~#
```

Figure 2: Ping to the Internet

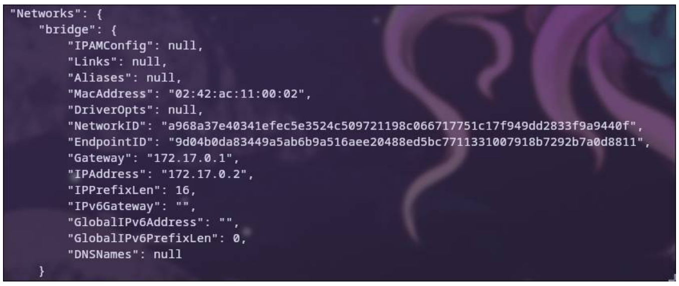


```
root@ab85eae5b992:~# ping 10.0.0.21
PING 10.0.0.21 (10.0.0.21) 56(84) bytes of data.
64 bytes from 10.0.0.21: icmp_seq=1 ttl=63 time=0.789 ms
64 bytes from 10.0.0.21: icmp_seq=2 ttl=63 time=0.807 ms
64 bytes from 10.0.0.21: icmp_seq=3 ttl=63 time=0.749 ms
64 bytes from 10.0.0.21: icmp_seq=4 ttl=63 time=0.861 ms
64 bytes from 10.0.0.21: icmp_seq=5 ttl=63 time=0.701 ms
64 bytes from 10.0.0.21: icmp_seq=6 ttl=63 time=0.946 ms
64 bytes from 10.0.0.21: icmp_seq=7 ttl=63 time=0.675 ms
64 bytes from 10.0.0.21: icmp_seq=8 ttl=63 time=0.510 ms
64 bytes from 10.0.0.21: icmp_seq=9 ttl=63 time=0.482 ms
64 bytes from 10.0.0.21: icmp_seq=10 ttl=63 time=1.16 ms
64 bytes from 10.0.0.21: icmp_seq=11 ttl=63 time=0.915 ms
64 bytes from 10.0.0.21: icmp_seq=12 ttl=63 time=0.708 ms
^C
--- 10.0.0.21 ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 11143ms
rtt min/avg/max/mdev = 0.482/0.775/1.161/0.178 ms
root@ab85eae5b992:~#
```

Figure 3: Ping the local machine

3.2.1 It works, but why?

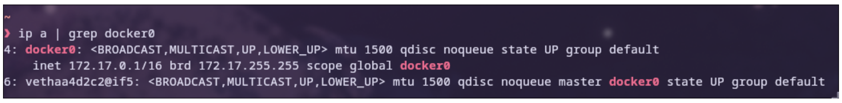
If we inspect our container using `docker inspect container-name`, we see that its IP is different from that of the lan.



```
"Networks": {
    "bridge": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": null,
        "MacAddress": "02:42:ac:11:00:02",
        "DriverOpts": null,
        "NetworkID": "a968a37e40341efec5e3524c509721198c066717751c17f949dd2833f9a9440f",
        "EndpointID": "9d04b0da83449a5ab6b9a516aee20488ed5bc7711331007918b7292b7a0d8811",
        "Gateway": "172.17.0.1",
        "IPAddress": "172.17.0.2",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "DNSNames": null
    }
```

Figure 4: docker inspect

This happens because when you install Docker, it creates a virtual interface `docker0` that is used as a network bridge to allow the container to communicate with the Internet and LAN. [7] There are other types of Docker networks, but they are not relevant for this exercise.[7]



```
~
❯ ip a | grep docker0
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
6: vethaa4d2c2@if5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
```

Figure 5: ip a | grep docker0

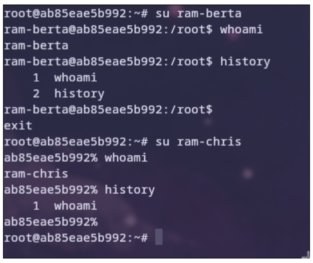
3.3 creating and managing users

To add groups and users and add those to the groups, use the commands `groupadd, useradd, usermod`. 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 log in as another user, use the `su` command.



```
root@ab85eae5b992:~# su ram-berta
ram-berta@ab85eae5b992:/root$ whoami
ram-berta
ram-berta@ab85eae5b992:/root$ history
    1  whoami
    2  history
ram-berta@ab85eae5b992:/root$
exit
root@ab85eae5b992:~# su ram-chris
ab85eae5b992% whoami
ram-chris
ab85eae5b992% history
    1  whoami
ab85eae5b992%
root@ab85eae5b992:~#
```

Figure 6: Login as Berta and Chris

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

3.4 Set directory privileges

The directories are created with this command and the `-p` stands for parent and creates parent directories if needed. For example, `mkdir /test/test2` wouldn't work if you don't have `/test`, but using `mkdir -p` instead will create `/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
```

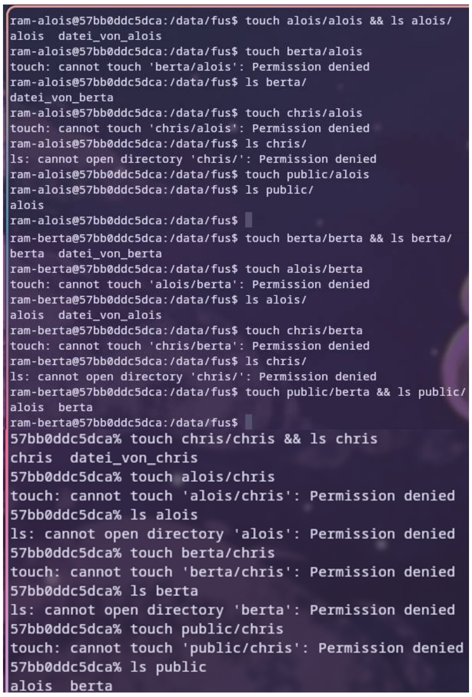
Three tools are used to set the permission: `chrgrp`, `chown` and `chmod`.
First, we want everyone in the group to have access to the directory for which `chgrp -R ram-Users /data/fus/` is used with the `-R` argument, which means recursive [12, 2].
To give everyone all the permissions in their own directory, we need to make them the owner of it using `chown -R username:groupname /data/fus/name-of-directory`.[10]
Now we can assign permissions to each directory using the `chmod`[8] command.
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
```

[8] Now, let us use this to set up the permissions accordingly

```
#giving the [g]roup [r]ead and [w]write 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/
```

[8]



```
ram-alois@57bb0ddc5dca:/data/fus$ touch alois/alois && ls alois/
alois  datei_von_alois
ram-alois@57bb0ddc5dca:/data/fus$ touch berta/alois
touch: cannot touch 'berta/alois': Permission denied
ram-alois@57bb0ddc5dca:/data/fus$ ls berta/
datei_von_berta
ram-alois@57bb0ddc5dca:/data/fus$ touch chris/alois
touch: cannot touch 'chris/alois': Permission denied
ram-alois@57bb0ddc5dca:/data/fus$ ls chris/
ls: cannot open directory 'chris/': Permission denied
ram-alois@57bb0ddc5dca:/data/fus$ touch public/alois
ram-alois@57bb0ddc5dca:/data/fus$ ls public/
alois
ram-alois@57bb0ddc5dca:/data/fus$
ram-berta@57bb0ddc5dca:/data/fus$ touch berta/berta && ls berta/
berta  datei_von_berta
ram-berta@57bb0ddc5dca:/data/fus$ touch alois/berta
touch: cannot touch 'alois/berta': Permission denied
ram-berta@57bb0ddc5dca:/data/fus$ ls alois/
alois  datei_von_alois
ram-berta@57bb0ddc5dca:/data/fus$ touch chris/berta
touch: cannot touch 'chris/berta': Permission denied
ram-berta@57bb0ddc5dca:/data/fus$ ls chris/
ls: cannot open directory 'chris/': Permission denied
ram-berta@57bb0ddc5dca:/data/fus$ touch public/berta && ls public/
alois  berta
ram-berta@57bb0ddc5dca:/data/fus$
57bb0ddc5dca% touch chris/chris && ls chris
chris  datei_von_chris
57bb0ddc5dca% touch alois/chris
touch: cannot touch 'alois/chris': Permission denied
57bb0ddc5dca% ls alois
ls: cannot open directory 'alois': Permission denied
57bb0ddc5dca% touch berta/chris
touch: cannot touch 'berta/chris': Permission denied
57bb0ddc5dca% ls berta
ls: cannot open directory 'berta': Permission denied
57bb0ddc5dca% touch public/chris
touch: cannot touch 'public/chris': Permission denied
57bb0ddc5dca% ls public
alois  berta
```

Figure 7: Testing permissions

If we log in as the users, we can see that everything is working as intended.

3.5 setting up ssh

The two new users required for this have already been created above in 3.3
To set up an ssh server we need to install the package, if we just add `ssh` to our install command in the Dockerfile we find out that this command 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
```

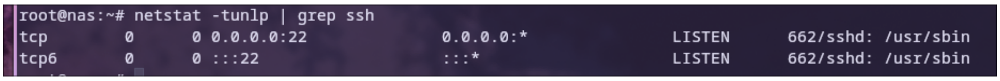
Now ssh is installed, but it needs to be started, all we need to do is edit the last line of the file to start the service as well.

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

To find out what port the server is listening on for ssh, we use the netstat command that comes with the `net-tools` package that we installed earlier. This is done with the command `netstat -tunlp | grep ssh`. The options of the command are explained below.

```
-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
```

[9]



```
root@nas:~# netstat -tunlp | grep ssh
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      662/sshd: /usr/sbin
tcp6       0      0 :::22                   :::*                    LISTEN      662/sshd: /usr/sbin
```

Figure 8: Search for port with netstat

Apparently it is a ”good practice” to switch from the default ssh port to a different port to avoid bots and script kiddies that scan the internet for public servers with ssh and test default passwords. I think this is snake oil to change ports for better security, because if you disable password authentication, have a strong password, or ban failing ips with tools like fail2ban, all the problems are solved anyway.[11]
For this we need to edit the file `/etc/ssh/sshd_config`.
I still changed the port to show how it would be done anyway. To do this, we can use the preinstalled text editor `sed`, so edit the file with the following command to change the port in the Dockerfile.

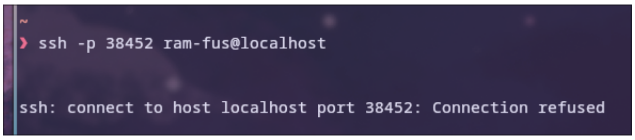
```
#-i edit the file in place without printing it to the console
#s to use the substitute command of sed
#'/s/string-you-want-to-replace/string-you-want-to-replace-it-with'
#/etc/ssh/sshd_config file that you want to edit
RUN sed -i 's/#Port 22/Port 38452/' /etc/ssh/sshd_config
```

When we try to ssh in with the created user, we cannot yet, since we have not published any ports in our container yet.

3.5.1 Logging On to the SSH Server

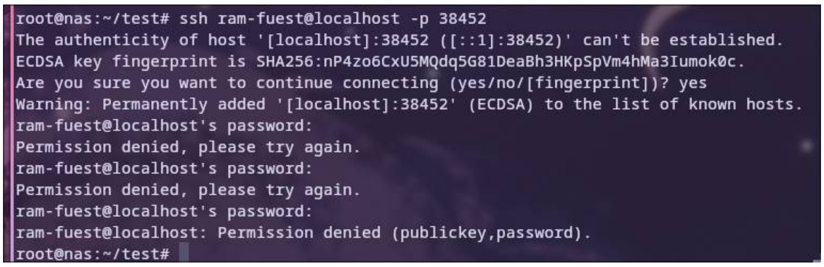
To do this, we need to add a line to the Dockerfile and edit the docker run command.

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



```
~
❯ ssh -p 38452 ram-fus@localhost

ssh: connect to host localhost port 38452: Connection refused
```

Figure 9: Connection refused



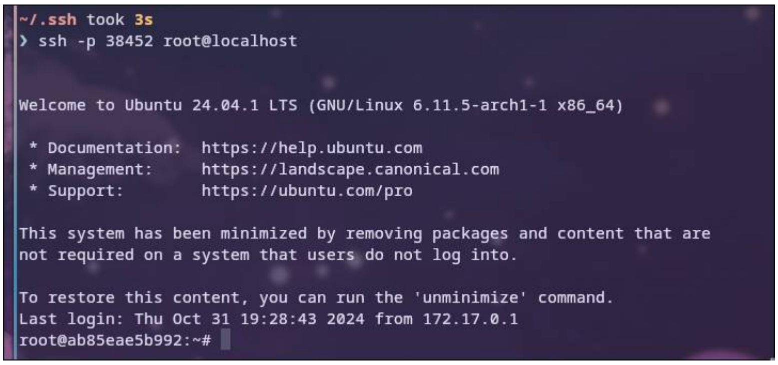
```
root@nas:~/test# ssh ram-fuest@localhost -p 38452
The authenticity of host '[localhost]:38452 ([::1]:38452)' can't be established.
ECDSA key fingerprint is SHA256:nP4zo6CxU5MQdq5G81DeaBh3HKpSpVm4hMa3Iumok0c.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[localhost]:38452' (ECDSA) to the list of known hosts.
ram-fuest@localhost's password:
Permission denied, please try again.
ram-fuest@localhost's password:
Permission denied, please try again.
ram-fuest@localhost's password:
ram-fuest@localhost: Permission denied (publickey,password).
root@nas:~/test#
```

Figure 10: Logging in without a password

Even if we log in now, it still won't work because the user doesn'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 because we do not have `sudo` setup, and having to type sudo for every command when we are the only user is both unnecessary and annoying.

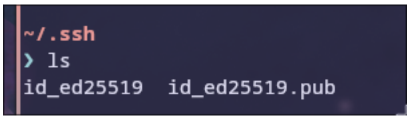


```
~/.ssh took 3s
❯ ssh -p 38452 root@localhost


Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.11.5-arch1-1 x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Thu Oct 31 19:28:43 2024 from 172.17.0.1
root@ab85eae5b992:~#
```

Figure 11: working login

3.5.2 enabeling keypair authenthication

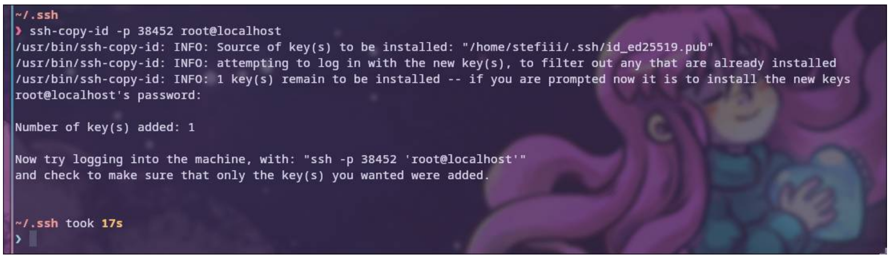
To generate a key pair, we go back to our host system and run the command `ssh-keygen -b 4096` to generate a 4096-bit SSH key. `ssh-keygen` On Linux, the keys are stored in the `/.ssh` directory, but you can specify a location with `-f`. The file that ends with `.pub` is the public key, and the other is the private key.



```
~/.ssh
❯ ls
id_ed25519  id_ed25519.pub
```

Figure 12: keys in the directory

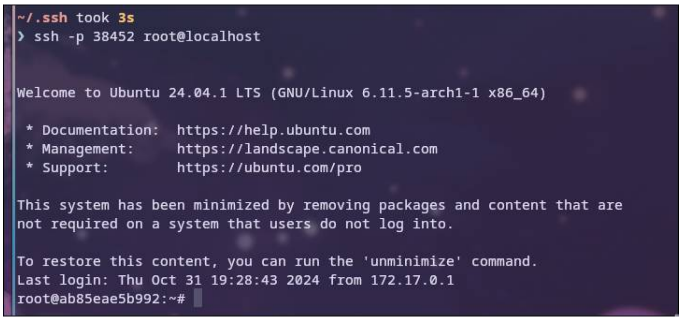
To copy the public key to the server we want to use it on, we use the command `ssh-copy-id` on Linux and `scp` on Windows and Mac. `ssh-copy-id`



```
~/.ssh
❯ ssh-copy-id -p 38452 root@localhost
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/stefiii/.ssh/id_ed25519.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@localhost's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh -p 38452 'root@localhost'"
and check to make sure that only the key(s) you wanted were added.


~/.ssh took 17s
❯
```

Figure 13: ssh-copy-id

After this we will not need to enter a password to authenticate.



```
~/.ssh took 3s
❯ ssh -p 38452 root@localhost


Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.11.5-arch1-1 x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Thu Oct 31 19:28:43 2024 from 172.17.0.1
root@ab85eae5b992:~#
```

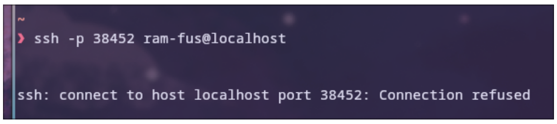
Figure 14: logging with a key

3.5.3 Disable password authentication

To only allow key authentication, we need to edit the `/etc/ssh/sshd_config` file again. To do this, we ssh into the server, open the file with a text editor of your choice, and edit this line.

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

If we try to log in as another user for which we do not have a key, we cannot connect.



```
~
❯ ssh -p 38452 ram-fus@localhost

ssh: connect to host localhost port 38452: Connection refused
```

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

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

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

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

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

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

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

4 List of figures

List of Figures

1 Grouplogo 1
2 Ping to the Internet 5
3 Ping the local machine 5
4 docker inspect 6
5 ip a | grep docker0 6
6 Login as Berta and Chris 7
7 Testing permissions 8
8 Search for port with netstat 9
9 Connection refused 10
10 Logging in without a password 10
11 working login 10
12 keys in the directory 11
13 ssh-copy-id 11
14 logging with a key 11
15 Not having a key 12

5 Attachments

Dockerfile

```
FROM ubuntu:latest

RUN apt update && apt upgrade -y &&\
    apt install iproute2 iputils-ping zsh net-tools vim -y
RUN ln -fs /usr/share/zoneinfo/Europe/Vienna /etc/localtime
RUN DEBIAN_FRONTEND=noninteractive apt install -y tzdata ssh
RUN echo 'root:password' | chpasswd
RUN sed -i 's/#Port 22/Port 38452/' /etc/ssh/sshd_config
RUN sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/'\
    /etc/ssh/sshd_config
RUN groupadd -g 324 ram-Users &&\
    useradd -u 1024 ram-alois &&\
    useradd -u 1124 ram-berta &&\
    useradd -u 1224 ram-chris &&\
    useradd ram-fus &&\
    useradd ram-ram
RUN usermod -g ram-Users ram-alois &&\
    usermod -g ram-Users ram-berta
RUN usermod --shell /bin/bash ram-alois &&\
    usermod --shell /bin/bash ram-berta &&\
    usermod --shell /bin/zsh ram-chris
RUN mkdir -p /data/fus &&\
    mkdir /data/fus/alois &&\
    mkdir /data/fus/berta &&\
    mkdir /data/fus/chris &&\
    mkdir /data/fus/public
RUN chgrp -R ram-Users /data/fus/ &&\
    chmod g+rw /data/fus/ &&\
    chown -R ram-alois:ram-Users /data/fus/alois/ &&\
    chmod -R u+wrx,g=r,o= /data/fus/alois/ &&\
    chown -R ram-berta:ram-Users /data/fus/berta/ &&\
    chmod -R u+wrx,g=r,o= /data/fus/berta/ &&\
    chown -R ram-chris:ram-Users /data/fus/chris/ &&\
    chmod -R u+wrx,g=,o= /data/fus/chris/ &&\
    chmod -R u+wrx,g+wrx,o=r /data/fus/public/
EXPOSE 38452
CMD service ssh start && tail -F /dev/null
```

alias.sh

```
#!/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'"
```