- Score

LCFS Simulator

https://killer.sh

Instructions

1. You have access to 5 servers:

- `terminal` *(default)*
- `web-srv1`
- `app-srv1`
- `data-001`
- `data-002`

2. The server on which to solve a question is mentioned in the question text. If **no server is mentioned** you'll need to create your solution on the default `terminal`
3. If you're asked to create solution files at `/opt/course/*`, then **always do this** on your main `terminal`
4. You can connect to each server using ssh, like `ssh web-srv1`
5. All server addresses are configured in `/etc/hosts` on each server
6. Nested ssh is **not possible**: you can only connect to each server from your main `terminal`
7. It's not possible to restart single servers. If deeper issues or misconfigurations occurred then the only solution might be to restart the complete simulator. This is possible using top menu by selecting "Restart Session"
8. This simulator might not contain all LFCS exam topics. Attendees are still required to learn and study the complete curriculum

> ***NOTE:*** Please let us know on Support or Slack if you find any issues, mistakes, spelling mistakes, unclear explanations or improvement suggestions etc. We appreciate any feedback that can help improving this simulator!

Question 1 | Kernel and System Info

Task weight: 2%

Write the Linux Kernel release into `/opt/course/1/kernel`.

Write the current value of Kernel parameter `ip_forward` into `/opt/course/1/ip_forward`.

Write the system timezone into `/opt/course/1/timezone`.

> ***NOTE:*** If no server is mentioned in the question text, you'll need to create your solution on the default `terminal`

Answer:

```
# find kernel version
➜ uname -r

# output value of kernel parameter
➜ cat /proc/sys/net/ipv4/ip_forward

# get current timezone
➜ date +%Z
cat /etc/timezone # also possible
```

The files should look like:

```
# /opt/course/1/kernel
5.15.0-69-generic
```

```
# /opt/course/1/ip_forward
1
```

```
# /opt/course/1/timezone
UTC
```

Question 2 | CronJobs

Task weight: 3%

On server `data-001`, user `asset-manager` is responsible for timed operations on existing data. Some changes and additions are necessary.

Currently there is one system-wide cronjob configured that runs every day at 8:30pm. Convert it from being a system-wide cronjob to one owned and executed by user `asset-manager`. This means that user should see it when running `crontab -l`.

Create a new cronjob owned and executed by user `asset-manager` that runs `bash /home/asset-manager/clean.sh` every week on Monday and Thursday at 11:15am.

> ***NOTE:*** You can connect to servers using ssh, for example `ssh data-001`

Answer:

Step 1

Here we should move a cronjob from system-wide to user `asset-manager`. First we check out that cronjob:

```
➜ ssh data-001

➜ root@data-001:~$ vim /etc/crontab
```

```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
17 *  * * * root    cd / && run-parts --report /etc/cron.hourly
25 6  * * * root  test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6  * * 7 root  test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6  1 * * root  test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
30 20 * * * root bash /home/asset-manager/generate.sh   # THAT'S THE ONE executed at 8:30pm
```

We go ahead and cut this line from that file, or copy it and remove it later! Next we're going to add it to the users cronjobs:

```
➜ root@data-001:~$ su asset-manager

➜ asset-manager@data-001:/root$ cd

➜ asset-manager@data-001:~$ pwd
/home/asset-manager

➜ asset-manager@data-001:~$ crontab -l # list cronjobs
no crontab for asset-manager

➜ asset-manager@data-001:~$ crontab -e # edit cronjobs
```

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
```

```
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
30 20 * * * bash /home/asset-manager/generate.sh   # MAKE SURE to remove the user
```

> ***NOTE:*** Here we shouldn't specify this any longer!

The system-wide cronjobs in `/etc/crontab` always specify the user that executes the command. Now it's no longer necessary.

After saving the file we should be able to:

```
➜ asset-manager@data-001:~$ crontab -l
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
30 20 * * * bash /home/asset-manager/generate.sh
```

Now we see that migrated cronjob!

Step 2

For the next step we should add a new cronjob. We can just copy and then change the existing one to our needs:

```
➜ asset-manager@data-001:~$ crontab -e
```

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
30 20 * * * bash /home/asset-manager/generate.sh
15 11 * * 1,4 bash /home/asset-manager/clean.sh   # the new one
```

Save and check:

```
➜ asset-manager@data-001:~$ crontab -l
...
30 20 * * * bash /home/asset-manager/generate.sh
15 11 * * 1,4 bash /home/asset-manager/clean.sh
```

For guidance check the comments in `/etc/crontab`, they're really useful. Instead of numbers for the days we can also use the actual names of days:

```
15 11 * * mon,thu bash /home/asset-manager/clean.sh   # also possible
```

The files for user crontabs are stored at location `/var/spool/cron/crontabs` and user root can access those.

Question 3 | Time synchronisation Configuration

Task weight: 3%

Time synchronisation configuration needs to be updated:

1. Set `0.pool.ntp.org` and `1.pool.ntp.org` as main NTP servers
2. Set `ntp.ubuntu.com` and `0.debian.pool.ntp.org` as fallback NTP servers
3. The maximum poll interval should be `1000` seconds and the connection retry `20` seconds

Answer:

> ***Info:*** Use `man timesyncd.conf` for help

A good idea would probably to take a look at the current situation:

```
→ timedatectl
               Local time: Sun 2023-06-11 16:29:05 UTC
           Universal time: Sun 2023-06-11 16:29:05 UTC
                 RTC time: Sun 2023-06-11 16:29:05
                Time zone: UTC (UTC, +0000)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

Here we see for example the current local time and timezone. Let's open the configuration:

```
→ sudo vim /etc/systemd/timesyncd.conf
```

```
[Time]
NTP=0.de.pool.ntp.org 1.de.pool.ntp.org 2.de.pool.ntp.org
```

We see three german NTP servers currently configured via setting `NTP`.

Test NTP servers

We can test single NTP servers manually for a sense of certainty:

```
→ ntpdate -q 0.de.pool.ntp.org # just query, don't update
server 85.215.93.134, stratum 2, offset -0.000523, delay 0.05086
server 85.214.46.39, stratum 3, offset +0.001502, delay 0.04944
server 129.70.132.32, stratum 2, offset -0.003332, delay 0.04881
server 141.82.25.202, stratum 2, offset -0.001288, delay 0.04404
11 Jun 15:50:41 ntpdate[3043]: adjust time server 141.82.25.202 offset -0.001288 sec

→ ntpdate -q www.google.de # that one won't work
11 Jun 15:49:40 ntpdate[3042]: no server suitable for synchronization found
```

Above we see one successful request and one to `www.google.de` that failed. This is correct because the Google web-domain doesn't provide a NTP service.

Step 1: Main servers

We adjust the config:

```
→ sudo vim /etc/systemd/timesyncd.conf
```

```
[Time]
NTP=0.pool.ntp.org 1.pool.ntp.org
```

Step 2: Fallback servers

Often times various settings are already included in the `timesyncd.conf` but commented out. Here it seems that we've to work with a pretty clean file. Hence we can use `man timesyncd.conf` for help:

```
[Time]
NTP=0.pool.ntp.org 1.pool.ntp.org
FallbackNTP=ntp.ubuntu.com 0.debian.pool.ntp.org
```

Step 3: Remaining settings

Here we also use the man pages as help:

```
[Time]
NTP=0.pool.ntp.org 1.pool.ntp.org
FallbackNTP=ntp.ubuntu.com 0.debian.pool.ntp.org
PollIntervalMaxSec=1000
ConnectionRetrySec=20
```

Final: Restart service

Now we restart the service:

```
➜ sudo service systemd-timesyncd restart
```

Good to check the service status for warnings or errors:

```
➜ sudo service systemd-timesyncd status
● systemd-timesyncd.service - Network Time Synchronization
     Loaded: loaded (/lib/systemd/system/systemd-timesyncd.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2023-07-27 15:40:11 UTC; 2s ago
       Docs: man:systemd-timesyncd.service(8)
   Main PID: 161213 (systemd-timesyn)
     Status: "Initial synchronization to time server 162.159.200.123:123 (0.pool.ntp.org)."
      Tasks: 2 (limit: 2234)
     Memory: 1.3M
        CPU: 100ms
     CGroup: /system.slice/systemd-timesyncd.service
             └─161213 /lib/systemd/systemd-timesyncd

Jul 27 15:40:11 terminal systemd[1]: Starting Network Time Synchronization...
Jul 27 15:40:11 terminal systemd[1]: Started Network Time Synchronization.
Jul 27 15:40:11 terminal systemd-timesyncd[161213]: Initial synchronization to time server 162.159.200.123:123
(0.pool.ntp.org).
```

Status output looking good. In the logs above we can see which NTP server was used for synchronisation. We could also check the logs with:

```
➜ sudo grep systemd-timesyncd /var/log/syslog
...
Jul 27 15:40:11 ubuntu2204 systemd[1]: systemd-timesyncd.service: Deactivated successfully.
Jul 27 15:40:11 ubuntu2204 systemd-timesyncd[161213]: Initial synchronization to time server 162.159.200.123:123
(0.pool.ntp.org).
```

Server `0.pool.ntp.org` was used here which means our configuration change worked.

Question 4 | Environment Variables

Task weight: 3%

There is an existing env variable for user `student@terminal`: `VARIABLE1=random-string`, defined in file `.bashrc`. Create a new script under `/opt/course/4/script.sh` which:

1. Defines a new env variable `VARIABLE2` with content `v2`, only available in the script itself
2. Outputs the content of the env variable `VARIABLE2`
3. Defines a new env variable `VARIABLE3` with content `${VARIABLE1}-extended`, available in the script itself and all child processes of the shell as well
4. Outputs the content of the env variable `VARIABLE3`

> ***NOTE:*** Do not alter the `.bashrc` file, everything needs to be done in the script itself

Answer:

Well, let's check the existing variable and its content as mentioned:

```
➜ echo $VARIABLE1
random-string

➜ env | grep VARIABLE
VARIABLE1=random-string
```

How the variable's value defined? Let's check the `.bashrc` file:

```
➜ cat .bashrc | grep VARIABLE1
export VARIABLE1=random-string
```

Now let's create a script which will define a new env variable called `VARIABLE2` with content `v2`:

```
➜ vim /opt/course/4/script.sh
```

```
VARIABLE2="v2"
echo $VARIABLE2
```

We give it a try, it should output the variable content, but shouldn't make it available (export) afterwards:

```
➜ bash /opt/course/4/script.sh
v2
```

Finally, we define the third environment variable called `VARIABLE3` within the same script

```
VARIABLE2="v2"
echo $VARIABLE2
export VARIABLE3="${VARIABLE1}-extended"    # add
echo $VARIABLE3                             # add
```

Run and check the result

```
➜ sh /opt/course/4/script.sh
v2
random-string-extended
```

What's the difference between export and not using export? Let's demonstrate it:

```
➜ TEST1=test1

➜ export TEST2=test2

➜ echo $TEST1
test1

➜ echo $TEST2
test2

➜ bash

➜ echo $TEST1 # variable NOT available in subprocesses

➜ echo $TEST2 # exported variable available in subprocesses
test2
```

Question 5 | Archives and Compression

Task weight: 6%

There is archive `/imports/import001.tar.bz2` on server `data-001`. You're asked to create a new gzip compressed archive with its raw contents. Make sure the original archive will remain untouched.

Store the new archive under `/imports/import001.tar.gz`. Compression should be the best possible, using gzip.

To make sure both archives contain the same files, write a list of their sorted contents into `/imports/import001.tar.bz2_list` and `/imports/import001.tar.gz_list`.

Answer:

We connect to `data-001` and have a look at the folder:

```
→ ssh data-001

→ root@data-001:~$ cd /imports

→ root@data-001:/imports$ ls -lh
total 1.5K
-rw-r--r-- 1 root root 560 Jul 16 13:49 import001.tar.bz2
```

Possibility 1 (use the tar layer)

We extract the `bzip2` archive and receive an uncompressed `tar` archive:

> ***INFO:*** We can install `bzip2` via the package manager if not available

```
→ root@data-001:/imports$ bunzip2 -k import001.tar.bz2

→ root@data-001:/imports$ ls -lh
total 3.0K
-rw-r--r-- 1 root root 20K Jul 16 14:20 import001.tar # combination of all files without compression
-rw-r--r-- 1 root root 550 Jul 16 14:20 import001.tar.bz2
```

Every tar archive contains a "tar" data layer. This can then be further compressed with various compression algorithms. Here we can now go ahead and create a new gzip compression from the tar layer:

```
→ root@data-001:/imports$ gzip --best import001.tar

→ root@data-001:/imports$ ls -lh
total 2.0K
-rw-r--r-- 1 root root 550 Jul 16 14:20 import001.tar.bz2
-rw-r--r-- 1 root root 544 Jul 16 14:20 import001.tar.gz
```

Possibility 2 (completely extract and pack again)

We extract the files into a new subfolder:

```
→ root@data-001:/imports$ mkdir import001

→ root@data-001:/imports$ tar xf import001.tar.bz2 -C import001

→ root@data-001:/imports$ find import001/
import001/
import001/2ba047d9-a9b3-4261-a4a0-0d23447ebdcd
import001/2ba047d9-a9b3-4261-a4a0-0d23447ebdcd/e48edbdd
import001/2ba047d9-a9b3-4261-a4a0-0d23447ebdcd/fc2639f1
import001/2ba047d9-a9b3-4261-a4a0-0d23447ebdcd/8b718f8f
import001/2ba047d9-a9b3-4261-a4a0-0d23447ebdcd/5d517b37
import001/5d517b37-efd3-4872-b107-502aa4b58b4c
...
```

Now we create the new required archive

```
→ root@data-001:/imports$ GZIP=-9 tar czf import001.tar.gz -C import001 .
```

Using the GZIP env variable is deprecated, instead we could use:

```
→ root@data-001:/imports$ tar -I 'gzip -9' -cf import001.tar.gz -C import001 .
```

We should see:

```
→ root@data-001:/imports$ ls -lh
total 4.5K
drwxr-xr-x 7 ubuntu ubuntu   7 Jul 16 13:32 import001
-rw-r--r-- 1 root   root   550 Jul 16 13:57 import001.tar.bz2
-rw-r--r-- 1 root   root   531 Jul 16 14:00 import001.tar.gz
```

Finally

We ensure that both archives contain the same files and structure:

```
→ root@data-001:/imports$ tar tf import001.tar.bz2 | sort > import001.tar.bz2_list

→ root@data-001:/imports$ tar tf import001.tar.gz | sort > import001.tar.gz_list
```

To compare further we could use `cat import001.tar.bz2_list | sha512sum` and compare the hashes.

To see some info about the compression ratio we can run

```
➜ root@data-001:/imports$ gzip -l import001.tar.gz
```

Finally we should have these files:

```
➜ root@data-001:/imports$ rm -rf import001

➜ root@data-001:/imports$ ls -lha
total 6.0K
drwxr-xr-x 2 root root    6 Jul 16 14:07 .
drwxr-xr-x 3 root root    3 Jul 16 13:28 ..
-rw-r--r-- 1 root root  550 Jul 16 13:57 import001.tar.bz2
-rw-r--r-- 1 root root 1.2K Jul 16 14:07 import001.tar.bz2_list
-rw-r--r-- 1 root root  531 Jul 16 14:00 import001.tar.gz
-rw-r--r-- 1 root root 1.2K Jul 16 14:07 import001.tar.gz_list
```

Question 6 | User, Groups and Sudoers

Task weight: 5%

On server `app-srv1`:

1. Change the primary group of user `user1` to `dev` and the home directory to `/home/accounts/user1`
2. Add a new user `user2` with groups `dev` and `op`, home directory `/home/accounts/user2`, terminal `/bin/bash`
3. User `user2` should be able to execute `sudo bash /root/dangerous.sh` without having to enter the root password

Answer:

Step 1

We can use different approaches. We could:

```
➜ ssh app-srv1

➜ root@app-srv1:~$ usermod -d /home/accounts/user1 user1
```

Or we could edit `/etc/passwd` manually:

```
➜ root@app-srv1:~$ vim /etc/passwd
```

```
root:x:0:0:root:/root:/bin/bash
...
lxd:x:999:100::/var/snap/lxd/common/lxd:/bin/false
ntp:x:113:121::/nonexistent:/usr/sbin/nologin
user1:x:1001:1001::/home/accounts/user1:/bin/bash # change path
```

No matter what solution, this should be correct:

```
➜ root@app-srv1:~$ su user1

➜ user1@app-srv1:/root$ cd

➜ user1@app-srv1:~$ pwd
/home/accounts/user1
```

And to change the primary group:

```
➜ root@app-srv1:~$ usermod -g dev user1

➜ root@app-srv1:~$ groups user1
user1 : dev
```

Step 2

First we can check available options:

```
➜ root@app-srv1:~$ useradd -h
Usage: useradd [options] LOGIN
       useradd -D
       useradd -D [options]

Options:
      --badnames                do not check for bad names
  -b, --base-dir BASE_DIR       base directory for the home directory of the
```

```
                            new account
    --btrfs-subvolume-home  use BTRFS subvolume for home directory
-c, --comment COMMENT       GECOS field of the new account
-d, --home-dir HOME_DIR     home directory of the new account                  # useful
-D, --defaults              print or change default useradd configuration
-e, --expiredate EXPIRE_DATE expiration date of the new account
-f, --inactive INACTIVE     password inactivity period of the new account
-g, --gid GROUP             name or ID of the primary group of the new
                            account
-G, --groups GROUPS         list of supplementary groups of the new            # useful
                            account
-h, --help                  display this help message and exit
-k, --skel SKEL_DIR         use this alternative skeleton directory
-K, --key KEY=VALUE         override /etc/login.defs defaults
-l, --no-log-init           do not add the user to the lastlog and
                            faillog databases
-m, --create-home           create the users home directory                     # useful
-M, --no-create-home        do not create the user's home directory
-N, --no-user-group         do not create a group with the same name as
                            the user
-o, --non-unique            allow to create users with duplicate
                            (non-unique) UID
-p, --password PASSWORD     encrypted password of the new account
-r, --system                create a system account
-R, --root CHROOT_DIR       directory to chroot into
-P, --prefix PREFIX_DIR     prefix directory where are located the /etc/* files
-s, --shell SHELL           login shell of the new account                     # useful
-u, --uid UID               user ID of the new account
-U, --user-group            create a group with the same name as the user
-Z, --selinux-user SEUSER   use a specific SEUSER for the SELinux user mapping
    --extrausers            Use the extra users database
```

Using the correct arguments we create the required new user:

```
➜ root@app-srv1:~$ useradd -s /bin/bash -m -d /home/accounts/user2 -G dev,op user2
```

To verify that it was added to the required groups:

```
➜ root@app-srv1:~$ cat /etc/group | grep user2
op:x:1003:user2
dev:x:1004:user2
user2:x:1005:
```

Step 3

Now it's getting interesting. We can try to execute the script with current configuration:

```
➜ root@app-srv1:~$ su user2

➜ user2@app-srv1:/root$ cd

➜ user2@app-srv1:~$ bash /root/dangerous.sh
bash: /root/dangerous.sh: Permission denied # DENIED
```

We need to configure sudoers to allow this specific script call. We should always edit the `/etc/sudoers` file using the command `visudo`, because it performs proper syntax validation before saving the file. Any misconfiguration of that file could lock us out of the system for good. So we do as root:

```
➜ root@app-srv1:~$ visudo
```

```
...

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "@include" directives:

@includedir /etc/sudoers.d

user2 ALL=(root) NOPASSWD: /bin/bash /root/dangerous.sh # ADD THIS
```

You can exit via `Ctrl + X`, then `Y` and then `Enter` to save.

And to verify:

```
➜ user2@app-srv1:~$ sudo bash /root/dangerous.sh
Sun Jun 11 17:54:20 UTC 2023
dangerous
```

Question 7 | Network Packet Filtering

Task weight: 8%

Server `data-002` is used for big data and provides internally used apis for various data operations. You're asked to implement network packet filters on interface `eth0` on `data-002`:

1. Port `5000` should be closed
2. Redirect all traffic on port `6000` to local port `6001`
3. Port `6002` should only be accessible from IP `192.168.10.80` (server `data-001`)
4. Block all outgoing traffic to IP `192.168.10.70` (server `app-srv1`)

> ***NOTE:*** In case of misconfiguration you can still access the instance using `sudo lxc exec data-002 bash`

Answer:

First we could test the mentioned ports on `data-002` from remote:

```
➜ curl data-002:5000
app on port 5000

➜ curl data-002:6000
curl: (7) Failed to connect to data-002 port 6000 after 4 ms: Connection refused

➜ curl data-002:6001
app on port 6001

➜ curl data-002:6002
app on port 6002
```

Further we can check for existing iptables rules and interfaces, because we're asked to implement the filters for `eth0`:

```
➜ ssh data-002

➜ root@data-002:~$ iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

➜ root@data-002:~$ iptables -L -t nat
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination

➜ root@data-002:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
26: eth0@if27: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 00:16:3e:6c:54:78 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.10.90/24 metric 100 brd 192.168.10.255 scope global dynamic eth0
       valid_lft 2752sec preferred_lft 2752sec
    inet6 fd42:a4f:8f61:21e3:216:3eff:fe6c:5478/64 scope global dynamic mngtmpaddr noprefixroute
       valid_lft 3305sec preferred_lft 3305sec
    inet6 fe80::216:3eff:fe6c:5478/64 scope link
```

```
       valid_lft forever preferred_lft forever
```

Above we can see the `etc0` interface and that there are no existing iptables rules implemented.

Step 1

We're asked to close port `5000` and are going to use iptables for it:

```
➜ root@data-002:~$ iptables -A INPUT -i eth0 -p tcp --dport 5000 -j DROP

➜ root@data-002:~$ iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  anywhere             anywhere             tcp dpt:5000 # new rule

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Done, let's give it a test:

```
➜ root@data-002:~$ curl localhost:5000   # still works on localhost
app on port 5000

➜ root@data-002:~$ exit

➜ curl data-001:5000                     # blocked from remote
curl: (7) Failed to connect to data-001 port 5000 after 0 ms: Connection refused
```

Step 2

Now we're going to perform some NAT for connections on port `6000`:

```
➜ root@data-002:~$ iptables -A PREROUTING -i eth0 -t nat -p tcp --dport 6000 -j REDIRECT --to-port 6001
```

View existing NAT rules:

```
➜ root@data-002:~$ iptables -L -t nat
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
REDIRECT   tcp  --  anywhere             anywhere             tcp dpt:x11 redir ports 6001 # new rule

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
```

If we would like to clear these we could run `iptables -F -t nat`. Let's test the result:

```
➜ root@data-002:~$ curl localhost:6000 # fail
curl: (7) Failed to connect to localhost port 6000 after 0 ms: Connection refused

➜ root@data-002:~$ exit

➜ curl data-002:6000 # fail
app on port 6001
```

Above we see the redirect from `6000` to `6001` works.

Step 3

Now we're asked to open port `5000` only from a specific source IP:

```
➜ root@data-002:~$ iptables -A INPUT -i eth0 -p tcp --dport 6002 -s 192.168.10.80 -j ACCEPT

➜ root@data-002:~$ iptables -A INPUT -i eth0 -p tcp --dport 6002 -j DROP
```

The idea there is that we first allow that IP and then deny all other traffic on that port. Let's verify it:

```
→ curl data-002:6002
^C # timeout

→ ssh data-001

→ root@data-001:~$ curl data-002:6002
app on port 6002 # success
```

Step 4

In the final step we need to drop outgoing packages:

```
→ root@data-002:~$ nc app-srv1 22
SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.1
^C  # success

→ root@data-002:~$ iptables -A OUTPUT -d 192.168.10.70 -p tcp -j DROP

→ root@data-002:~$ nc app-srv1 22
^C  # timeout

→ root@data-002:~$ nc data-001 22
SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.1
^C  # success
```

Above we can see that outgoing connections to `app-srv1` no longer work and they time out.

Question 8 | Disk Management

Task weight: 4%

Your team selected you for this task because of your deep filesystem and disk/devices expertise. Solve the following steps to not let your team down:

1. Format `/dev/vdb` with `ext4`, mount it to `/mnt/backup-black` and create empty file `/mnt/backup-black/completed`
2. Find the fullest of the disks `/dev/vdc` and `/dev/vdd`. Then empty the `.trash` folder on it
3. There are two processes running: `dark-matter-v1` and `dark-matter-v2`. Find the one that consumes more memory. Then unmount the disk where the process executable is located on

Answer:

A good way to start is probably to list existing disks:

```
→ sudo fdisk -l
...
Disk /dev/vdb: 100 MiB, 104857600 bytes, 204800 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/vdc: 100 MiB, 104857600 bytes, 204800 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/vdd: 100 MiB, 104857600 bytes, 204800 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/vde: 100 MiB, 104857600 bytes, 204800 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/vdf: 100 MiB, 104857600 bytes, 204800 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
...
```

Another way with good list style and information:

```
➜ lsblk -f
NAME                      FSTYPE      FSVER    LABEL ...  FSAVAIL FSUSE% MOUNTPOINTS
...
vdb
vdc                       ext4        1.0            ...        0    98% /mnt/backup001
vdd                       ext4        1.0            ...     77.7M     6% /mnt/backup002
vde                       ext4        1.0            ...     82.6M     0% /mnt/app-8e127b55
vdf                       ext4        1.0            ...     82.6M     0% /mnt/app-4e9d7e1e
...
```

Or even another way:

```
➜ df -h
Filesystem                         Size  Used Avail Use% Mounted on
tmpfs                              198M  1.1M  197M   1% /run
/dev/mapper/ubuntu--vg-ubuntu--lv   62G  5.6G   54G  10% /
tmpfs                              988M     0  988M   0% /dev/shm
tmpfs                              5.0M     0  5.0M   0% /run/lock
/dev/vda2                          2.0G  130M  1.7G   8% /boot
tmpfs                              198M  4.0K  198M   1% /run/user/1000
/dev/vdc                            90M   51M   33M  61% /mnt/backup001
/dev/vdd                            90M  5.1M   78M   7% /mnt/backup002
/dev/vde                            90M   44K   83M   1% /mnt/app-8e127b55
/dev/vdf                            90M   44K   83M   1% /mnt/app-4e9d7e1e
```

Step 1

We go right ahead and format the disk:

```
➜ sudo mkfs -t ext4 /dev/vdb
mke2fs 1.46.5 (30-Dec-2021)
Creating filesystem with 25600 4k blocks and 25600 inodes

Allocating group tables: done
Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting information: done
```

Then we mount it at the required location and create the file:

```
➜ sudo mkdir /mnt/backup-black

➜ sudo mount /dev/vdb /mnt/backup-black

➜ sudo touch /mnt/backup-black/completed

➜ df -h
Filesystem                         Size  Used Avail Use% Mounted on
tmpfs                              198M  1.4M  197M   1% /run
/dev/mapper/ubuntu--vg-ubuntu--lv   62G  7.5G   52G  13% /
tmpfs                              988M     0  988M   0% /dev/shm
tmpfs                              5.0M     0  5.0M   0% /run/lock
/dev/vda2                          2.0G  130M  1.7G   8% /boot
tmpfs                              198M  4.0K  198M   1% /run/user/1000
tmpfs                              1.0M     0  1.0M   0% /var/snap/lxd/common/ns
/dev/vdc                            90M   51M   33M  61% /mnt/backup001
/dev/vdd                            90M  5.1M   78M   7% /mnt/backup002
/dev/vde                            90M   44K   83M   1% /mnt/app-8e127b55
/dev/vdf                            90M   44K   83M   1% /mnt/app-4e9d7e1e
/dev/vdb                            90M   24K   83M   1% /mnt/backup-black # this is us
tmpfs                              198M  4.0K  198M   1% /run/user/0
```

Step 2

Now we need to check the used disk space of two disks:

```
➜ df -h
Filesystem                         Size  Used Avail Use% Mounted on
tmpfs                              198M  1.4M  197M   1% /run
/dev/mapper/ubuntu--vg-ubuntu--lv   62G  7.5G   52G  13% /
tmpfs                              988M     0  988M   0% /dev/shm
tmpfs                              5.0M     0  5.0M   0% /run/lock
/dev/vda2                          2.0G  130M  1.7G   8% /boot
tmpfs                              198M  4.0K  198M   1% /run/user/1000
tmpfs                              1.0M     0  1.0M   0% /var/snap/lxd/common/ns
/dev/vdc                            90M   51M   33M  61% /mnt/backup001 # compare
/dev/vdd                            90M  5.1M   78M   7% /mnt/backup002 # compare
```

```
/dev/vde                        90M   44K   83M   1% /mnt/app-8e127b55
/dev/vdf                        90M   44K   83M   1% /mnt/app-4e9d7e1e
/dev/vdb                        90M   24K   83M   1% /mnt/backup-black
```

We see that we need to clear up disk space on `/mnt/backup001`:

```
➜ sudo rm -rf /mnt/backup001/.trash/*
```

The results should be shown:

```
➜ df -h
...
/dev/vdc                        90M   28K   83M   1% /mnt/backup001      # much space gained
/dev/vdd                        90M  5.1M   78M   7% /mnt/backup002
```

Step 3

For the final step we need to check the memory of two processes and then find out on which disk the largest consumer runs:

```
➜ top -b | grep dark-matter
   6203 root      20   0    7072   4244   3132 S   0.0   0.2   0:00.00 dark-matter-v1
   6204 root      20   0   16288  13460   3108 S   0.0   0.6   0:00.00 dark-matter-v2
```

We see both processes with their memory consumption in column 9. We can also use `ps` which can show the full path of the executables, which we also need:

```
➜ ps aux | grep dark-matter
root        6203  0.0  0.2   7072  4244 pts/2    S    17:17   0:00 /mnt/app-8e127b55/dark-matter-v1
root        6204  0.0  0.6  16288 13460 pts/2    S    17:17   0:00 /mnt/app-4e9d7e1e/dark-matter-v2
```

Above we see the memory usage in column 4 and the full paths at the very end. This means process `dark-matter-v2` is the offender and the executable is to be found at `/mnt/app-4e9d7e1e/dark-matter-v2`. So we need to do:

```
➜ df -h | grep /mnt/app-4e9d7e1e
/dev/vdf            90M   44K   83M   1% /mnt/app-4e9d7e1e     # to see the disk and mount point

➜ sudo umount /mnt/app-4e9d7e1e
umount: /mnt/app-4e9d7e1e: target is busy.
```

The disk is busy, probably because of that one process! But we can check for any other processes blocking this:

```
➜ sudo lsof | grep /mnt/app-4e9d7e1e
dark-matt 6204    root  txt       REG             252,80    16488         12 /mnt/app-4e9d7e1e/dark-matter-v2
```

Well, only that one bad process! Hence we can finish this question with:

```
➜ sudo kill 6204 # your PID will be different

➜ sudo umount /mnt/app-4e9d7e1e
```

Question 9 | Find files with properties and perform actions

Task weight: 6%

There is a backup folder on server `data-001` at `/var/backup/backup-015`, it needs to be cleaned up.

First:

- Delete all files created before `01-01-2020`

Then for the remaining:

- Find all files smaller than `3KiB` and move these to `/var/backup/backup-015/small/`
- Find all files larger than `10KiB` and move these to `/var/backup/backup-015/large/`
- Find all files with permission `777` and move these to `/var/backup/backup-015/compromised/`

Answer:

First we find the backup location:

```
→ ssh data-001

→ root@data-001:~$ cd /var/backup/backup-015

→ root@data-001:/var/backup/backup-015$ ls | grep backup | wc -l
300
```

Seems to contain good amount of files! Now we need to clean it up. A good way for this is to use `find` with arguments and a command to execute, like:

```
find -exec echo {} \; # will find all files and runs "echo FILE" for each

find -exec echo {} +  # will find all files and runs "echo FILE1 FILE2 FILE3 ..."

man find # search for "exec" to see info and explanation

man find # search for "-newerXY" to find files "newer than date"
```

Delete files before date

Using this we can delete all files created before 2020. Always "debug" a command first by just listing without executing a command:

```
→ root@data-001:/var/backup/backup-015$ find ! -newermt "01/01/2020" -type f
...

→ root@data-001:/var/backup/backup-015$ find ! -newermt "01/01/2020" -type f | wc -l
17

→ root@data-001:/var/backup/backup-015$ find ! -newermt "01/01/2020" -type f -exec rm {} \;

→ root@data-001:/var/backup/backup-015$ ls | grep backup | wc -l
283 # 17 deleted
```

Move small files

Now we move all small files into the subfolder:

```
→ root@data-001:/var/backup/backup-015$ find -maxdepth 1 -size -3k -type f # find
...

→ root@data-001:/var/backup/backup-015$ find -maxdepth 1 -size -3k -type f | wc -l
28

→ root@data-001:/var/backup/backup-015$ find -maxdepth 1 -size -3k -type f -exec mv {} ./small \; # move
```

Move large files

Next we move all larger files into the subfolder:

```
→ root@data-001:/var/backup/backup-015$ find -maxdepth 1 -size +10k -type f # find
...

→ root@data-001:/var/backup/backup-015$ find -maxdepth 1 -size +10k -type f | wc -l
11

→ root@data-001:/var/backup/backup-015$ find -maxdepth 1 -size +10k -type f -exec mv {} ./large \; # move
```

Move open permission files

And finally we move all files with too open permissions into the subfolder:

```
→ root@data-001:/var/backup/backup-015$ find -maxdepth 1 -perm 777 -type f # find
...

→ root@data-001:/var/backup/backup-015$ find -maxdepth 1 -perm 777 -type f | wc -l
12

→ root@data-001:/var/backup/backup-015$ find -maxdepth 1 -perm 777 -type f -exec mv {} ./compromised \; # move
```

Result

```
➜ root@data-001:/var/backup/backup-015$ ls | grep backup | wc -l
232

➜ root@data-001:/var/backup/backup-015$ ls small/ | wc -l
28

➜ root@data-001:/var/backup/backup-015$ ls large/ | wc -l
11

➜ root@data-001:/var/backup/backup-015$ ls compromised/ | wc -l
12
```

Question 10 | SSHFS and NFS

Task weight: 7%

In this task it's required to access remote filesystems over network.

On your main server `terminal` use SSHFS to mount directory `/data-export` from server `app-srv1` to `/app-srv1/data-export`. The mount should be `read-write` and option `allow_other` should be enabled.

The NFS service has been installed on your main server `terminal`. Directory `/nfs/share` should be read-only accessible from `192.168.10.0/24`. On `app-srv1`, mount the NFS share `/nfs/share` to `/nfs/terminal/share`.

Answer:

SSHFS

We go ahead and create the SSHFS mount:

```
➜ man sshfs # always helpful

➜ sudo mkdir -p /app-srv1/data-export

➜ sudo sshfs -o allow_other,rw app-srv1:/data-export /app-srv1/data-export
```

No errors but also no output, we further verify:

```
➜ find /app-srv1/data-export
/app-srv1/data-export
/app-srv1/data-export/datRTG_1.gz
/app-srv1/data-export/datRTG_2.gz

➜ touch /app-srv1/data-export/new

➜ find /app-srv1/data-export
/app-srv1/data-export
/app-srv1/data-export/new
/app-srv1/data-export/datRTG_1.gz
/app-srv1/data-export/datRTG_2.gz

➜ ssh app-srv1 find /data-export
/data-export
/data-export/new
/data-export/datRTG_1.gz
/data-export/datRTG_2.gz
```

Above we can see that creating a new file works and that both directories are synced. Note that all users can now access that mount in read-write mode because of option allow_other, which might not be something we want in production environments!

NFS Server

We could start by verifying the service runs without issues:

```
➜ service --help                      # how to list all services?
Usage: service < option > | --status-all | [ service_name [ command | --full-restart ] ]

➜ service --status-all | grep nfs     # find nfs service
 [ - ]  nfs-common
 [ + ]  nfs-kernel-server

➜ service nfs-kernel-server status    # check nfs service status
● nfs-server.service - NFS server and services
     Loaded: loaded (/lib/systemd/system/nfs-server.service; enabled; vendor preset: enabled)
     Active: active (exited) since Thu 2023-06-15 15:09:18 UTC; 18min ago
   Main PID: 25267 (code=exited, status=0/SUCCESS)
        CPU: 8ms
```

NFS server seems to be running. We can expose certain directories via `/etc/exports`:

```
➜ sudo vim /etc/exports
```

```
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
/nfs/share            192.168.10.0/24(ro,sync,no_subtree_check,no_root_squash)
```

The file provides some comments with examples which can be very useful. After adding the exports we need to run:

```
➜ man exportfs # in case we forget the arguments: "-r" for reexport and "-a" for all

➜ sudo exportfs -ra

➜ showmount -e
Export list for terminal:
/nfs/share       192.168.10.0/24
```

NFS Client

Now we check if our NFS server-side settings can actually be accessed on client-side by NFS mounting:

```
➜ ssh app-srv1

➜ root@app-srv1:~$ nc -v terminal 2049 # for debugging we check can if NFS on port 2049 is open
Connection to terminal (192.168.100.2) 2049 port [tcp/nfs] succeeded!
^C

➜ root@app-srv1:~$ mkdir -p /nfs/terminal/share # we need to create the local mount destination directory

➜ root@app-srv1:~$ mount terminal:/nfs/share /nfs/terminal/share # mount the remote NFS export

➜ root@app-srv1:~$ find /nfs/terminal/share # view existing files
/nfs/terminal/share
/nfs/terminal/share/2.log
/nfs/terminal/share/3.log
/nfs/terminal/share/1.log

➜ root@app-srv1:~$ touch /nfs/terminal/share/new # not possible to create new file (because of read-only)
touch: cannot touch '/nfs/terminal/share/new': Read-only file system
```

Above we see that we were able to mount the required NFS export in read-only mode on `app-srv1`.

Question 11 | Docker Management

Task weight: 6%

Someone overheard that you're a Containerisation Specialist, so the following should be easy for you! Please:

1. Stop the Docker container named `frontend_v1`
2. Gather information from Docker container named `frontend_v2`:
   - Write its assigned ip address into `/opt/course/11/ip-address`
   - It has one volume mount. Write the volume mount destination directory into `/opt/course/11/mount-destination`

3. Start a new detached Docker container:
    - Name: `frontend_v3`
    - Image: `nginx:alpine`
    - Memory limit: `30m` (30 Megabytes)
    - TCP Port map: `1234/host` => `80/container`

Answer:

Dockerfile: list of commands from which an *Image* can be build

Image: binary file which includes all data/requirements to be run as a *Container*

Container: running instance of an *Image*

Registry: place where we can push/pull *Images* to/from

We first list all Docker containers:

```
➜ sudo docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS     NAMES
a9b334cfaae0   nginx:alpine   "/docker-entrypoint.…"   11 minutes ago   Up 11 minutes   80/tcp    frontend_v1
e68fa28f231d   nginx:alpine   "/docker-entrypoint.…"   7 minutes ago    Up 7 minutes    80/tcp    frontend_v2
```

Step 1

For the first step we stop the container:

```
➜ sudo docker stop frontend_v1
frontend_v1

➜ sudo docker ps -a  # show also stopped containers using "ps -a"
CONTAINER ID   IMAGE          ...  CREATED          STATUS                      PORTS     NAMES
e68fa28f231d   nginx:alpine   ...  8 minutes ago    Up 8 minutes                80/tcp    frontend_v2
a9b334cfaae0   nginx:alpine   ...  13 minutes ago   Exited (0) 45 seconds ago             frontend_v1
```

Step 2

Docker inspect provides the container configuration in JSON format which contains all information asked for in this task:

```
➜ sudo docker inspect frontend_v2
```

```
        "Mounts": [
            {
                "Type": "bind",
                "Source": "/var/www",
                "Destination": "/srv",                # WE NEED THIS
                "Mode": "",
                "RW": true,
                "Propagation": "rprivate"
            }
        ],
...
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID": "a550576248b3f6c15211c2ad10efc421c7b48285cd69d57b64d8dc7b1b59e0ef",
            "HairpinMode": false,
...
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "5889aae0b056e4f944d6fb9afd8edd26ecd589c5e45b7533b59c878138fb5625",
                    "EndpointID": "954fcc88c2b64ef9ba3809eb130060ceb064301c4906b4a7ab0902be6153eedb",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.3",         # WE NEED THIS
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:03",
                    "DriverOpts": null
                }
            }
        }
```

It's probably a good idea to search in the inspect output for specific values. For this we could open the output directly in vim:

```
➜ sudo docker inspect frontend_v2 | vim -
```

Now we only have to create the required files with their correct content (your container ip address might differ):

```
# /opt/course/11/ip-address
172.17.0.3
```

```
# /opt/course/11/mount-destination
/srv
```

Step 3

Finally we can start our own container! Unfortunately with very strict conditions to follow... so let's obey!

The help output for `docker run` usually provides all that's needed:

```
➜ sudo docker run --help

Usage:  docker run [OPTIONS] IMAGE [COMMAND] [ARG...]      # the order of arguments is important

Run a command in a new container

Options:
      --add-host list                Add a custom host-to-IP mapping (host:ip)
  -a, --attach list                  Attach to STDIN, STDOUT or STDERR
...
      --cpus decimal                 Number of CPUs
      --cpuset-cpus string           CPUs in which to allow execution (0-3, 0,1)
      --cpuset-mems string           MEMs in which to allow execution (0-3, 0,1)
  -d, --detach                       Run container in background and print container ID    # need this
      --detach-keys string           Override the key sequence for detaching a container
      --device list                  Add a host device to the container
      --device-cgroup-rule list      Add a rule to the cgroup allowed devices list
...
      --mount mount                  Attach a filesystem mount to the container
      --name string                  Assign a name to the container                        # need this
      --network network              Connect a container to a network
...
      --pids-limit int               Tune container pids limit (set -1 for unlimited)
      --platform string              Set platform if server is multi-platform capable
      --privileged                   Give extended privileges to this container
  -p, --publish list                 Publish a containers port(s) to the host              # need this
  -P, --publish-all                  Publish all exposed ports to random ports
...
```

Using this we can build the necessary run command:

```
➜ sudo docker run -d --name frontend_v3 --memory 30m -p 1234:80 nginx:alpine
1e7d4612df4aff82c96d2c65102966561038ebfb94a996c86afebf6e3cb1432a
```

In case the above command throws iptables errors we can restart the docker service:

```
➜ sudo docker rm --force frontend_v3 # delete existing container

➜ sudo service docker restart        # restart docker servier
```

Because of the port mapping we should now be able to do:

```
➜ curl localhost:1234
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Question 12 | Git Workflow

Task weight: 3%

You're asked to perform changes in the Git repository of the Auto-Verifier app:

1. Clone repository `/repositories/auto-verifier` to `/home/student/repositories/auto-verifier`.

   Perform the following steps in the newly cloned directory

2. Find the one of the branches `dev4`, `dev5` and `dev6` in which file `config.yaml` contains `user_registration_level: open`. Merge only that branch into branch `main`
3. In branch `main` create a new directory `logs` on top repository level. To ensure the directory will be committed create hidden empty file `.keep` in it
4. Commit your change with message `added log directory`

Answer:

Step 1: Clone Repository

Git is most often used to clone from and work with remote repositories on like Github or Gitlab. But most of Gits functionality can also be used locally. We go ahead and clone the local directory:

```
➜ git clone /repositories/auto-verifier ~/repositories/auto-verifier
Cloning into '/home/student/repositories/auto-verifier'...
done.

➜ cd ~/repositories/auto-verifier

➜ ~/repositories/auto-verifier$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Step 2: Find the correct branch

First we list all branches:

```
➜ ~/repositories/auto-verifier$ git branch
* main

➜ ~/repositories/auto-verifier$ git branch -a
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/dev4
  remotes/origin/dev5
  remotes/origin/dev6
  remotes/origin/main
```

We can simply move through all branches and check the file content:

```
➜ ~/repositories/auto-verifier$ git checkout dev4
Branch 'dev4' set up to track remote branch 'dev4' from 'origin'.
Switched to a new branch 'dev4'

➜ ~/repositories/auto-verifier$ grep user_registration_level config.yaml
user_registration_level: closed


➜ ~/repositories/auto-verifier$ git checkout dev5
Branch 'dev5' set up to track remote branch 'dev5' from 'origin'.
Switched to a new branch 'dev5'

➜ ~/repositories/auto-verifier$ grep user_registration_level config.yaml
user_registration_level: open

# Match! We could've stopped here, but let's also check the final branch

➜ ~/repositories/auto-verifier$ git checkout dev6
Branch 'dev6' set up to track remote branch 'dev6' from 'origin'.
Switched to a new branch 'dev6'

➜ ~/repositories/auto-verifier$ grep user_registration_level config.yaml
user_registration_level: closed
```

We could also check the commit that actually performed the change:

```
➜ ~/repositories/auto-verifier$ git checkout dev5
Branch 'dev5' set up to track remote branch 'dev5' from 'origin'.
Switched to a new branch 'dev5'

➜ ~/repositories/auto-verifier$ git log
commit cdf23b8c4000a4ff280f4feced059888e99d63e4 (HEAD -> dev5, origin/dev5)
Author: manager <manager@auto-verifier>
Date:   Tue Jan 10 17:03:55 2023 +1000

    updated config.yaml

commit 40842892dd0858eb96533654d5682c2c8b2ccb5c (origin/main, origin/HEAD, main)
Author: manager <manager@auto-verifier>
Date:   Fri Jan 6 06:33:43 2023 +1000

    set user_registration_level to closed

commit 9b73a28f2c87c6b34b9a779f5e82b4ebbf8bc78c
Author: manager <manager@auto-verifier>
Date:   Thu Jan 5 20:19:58 2023 +1000

    start of something cool

➜ ~/repositories/auto-verifier$ git diff main
diff --git a/config.yaml b/config.yaml
index bbbfb9b..5d1ccb4 100755
--- a/config.yaml
+++ b/config.yaml
@@ -31,7 +31,7 @@ system_transport_50x_list: 50x
 system_transport_50x_dsn: 'srv://default?list=50x'
 system_transport_50x_retry_strategy:
 system_transport_50x_ax_retries: 3
-user_registration_level: closed
+user_registration_level: open   # the change
 user_registration_enabled: false
 user_free_labels_enabled: true
 user_email_verify_enabled: true
```

Another way could also be to compare from branch `main` without actually switching into another:

```
➜ ~/repositories/auto-verifier$ git branch -a
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/dev4
  remotes/origin/dev5
  remotes/origin/dev6
  remotes/origin/main

➜ ~/repositories/auto-verifier$ git diff origin/dev5
diff --git a/config.yaml b/config.yaml
index 5d1ccb4..bbbfb9b 100755
--- a/config.yaml
+++ b/config.yaml
@@ -31,7 +31,7 @@ system_transport_50x_list: 50x
 system_transport_50x_dsn: 'srv://default?list=50x'
 system_transport_50x_retry_strategy:
 system_transport_50x_ax_retries: 3
-user_registration_level: open
+user_registration_level: closed
 user_registration_enabled: false
 user_free_labels_enabled: true
 user_email_verify_enabled: true
```

Now we simply have to merge branch `dev5` into `main`:

```
➜ ~/repositories/auto-verifier$git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

➜ ~/repositories/auto-verifier$ git branch
  dev4
  dev5
  dev6
* main

➜ ~/repositories/auto-verifier$ git merge dev5
Updating 4084289..cdf23b8
Fast-forward
 config.yaml | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

➜ ~/repositories/auto-verifier$ grep user_registration_level config.yaml
```

```
user_registration_level: open
```

Step 3: Create new directory

We're asked to create new directory, let's see what happens if we try to commit it just like that:

```
➜ ~/repositories/auto-verifier$ mkdir logs

➜ ~/repositories/auto-verifier$ ls -l
total 20
-rwxrwxr-x 1 student student 2015 Jul 27 14:20 config.yaml
drwxrwxr-x 2 student student 4096 Jul 27 14:16 lib
drwxrwxr-x 2 student student 4096 Jul 27 14:29 logs
-rwxrwxr-x 1 student student  205 Jul 27 14:16 main.go
-rwxrwxr-x 1 student student  133 Jul 27 14:16 README.md

➜ ~/repositories/auto-verifier$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Above we see that `git status` doesn't list the directory at all because it's empty. This means that it wouldn't be included in commits too. Now we add the requested file and see if things change:

```
➜ ~/repositories/auto-verifier$ touch logs/.keep

➜ ~/repositories/auto-verifier$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        logs/

nothing added to commit but untracked files present (use "git add" to track)
```

That looks better!

Step 4: Commit

Always best to confirm what's going to be committed:

```
➜ ~/repositories/auto-verifier$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        logs/

➜ ~/repositories/auto-verifier$ git add logs

➜ ~/repositories/auto-verifier$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   logs/.keep

➜ ~/repositories/auto-verifier$ git commit -m 'added log directory'
[main 3cc53ed] added log directory
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 logs/.keep
```

Question 13 | Runtime Security of processes

Task weight: 5%

There was a security alert which you need to follow up on. On server `web-srv1` there are three processes: `collector1`, `collector2`, and `collector3`. It was alerted that any of these might run periodically the per custom policy forbidden syscall `kill`.

End the process and remove the executable for those where this is true.

> ***INFO:*** You can use `strace -p PID`

Answer:

We should check the server for the mentioned processes:

```
➜ ssh web-srv1

➜ root@web-srv1:~$ ps aux | grep collector
root        3611  0.0  0.0 101924   624 ?        Sl   13:23   0:00 /bin/collector1
root        3612  0.0  0.0 101916   612 ?        Sl   13:23   0:00 /bin/collector2
root        3613  0.0  0.0 101928   616 ?        Sl   13:23   0:00 /bin/collector3
```

Now we investigate the kernel syscalls of `collector1`:

```
➜ root@web-srv1:~$ strace -p 3611 # your PID will be different!
restart_syscall(<... resuming interrupted read ...>) = -1 ETIMEDOUT (Connection timed out)
futex(0x4d4bb0, FUTEX_WAKE_PRIVATE, 1)  = 1
futex(0xc0000324c8, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x4d7460, FUTEX_WAIT_PRIVATE, 0, {tv_sec=0, tv_nsec=999999757}) = -1 ETIMEDOUT (Connection timed out)
futex(0x4d4bb0, FUTEX_WAKE_PRIVATE, 1)  = 1
futex(0xc0000324c8, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x4d7460, FUTEX_WAIT_PRIVATE, 0, {tv_sec=0, tv_nsec=999999756}) = -1 ETIMEDOUT (Connection timed out)
futex(0x4d4bb0, FUTEX_WAKE_PRIVATE, 1)  = 1
futex(0xc0000324c8, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x4d7460, FUTEX_WAIT_PRIVATE, 0, {tv_sec=0, tv_nsec=999999699}) = -1 ETIMEDOUT (Connection timed out)
futex(0x4d4bb0, FUTEX_WAKE_PRIVATE, 1)  = 1
futex(0xc0000324c8, FUTEX_WAKE_PRIVATE, 1) = 1
...
```

After watching for a while, there don't seem to be any kill syscalls. Next one is `collector2`:

```
➜ root@web-srv1:~$ strace -p 3612
restart_syscall(<... resuming interrupted read ...>) = -1 ETIMEDOUT (Connection timed out)
futex(0x4d2db0, FUTEX_WAKE_PRIVATE, 1)  = 1
kill(666, SIGTERM)                      = -1 ESRCH (No such process)
futex(0xc0000324c8, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x4d5660, FUTEX_WAIT_PRIVATE, 0, {tv_sec=0, tv_nsec=999999449}) = -1 ETIMEDOUT (Connection timed out)
futex(0x4d2db0, FUTEX_WAKE_PRIVATE, 1)  = 1
kill(666, SIGTERM)                      = -1 ESRCH (No such process)
futex(0xc0000324c8, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x4d5660, FUTEX_WAIT_PRIVATE, 0, {tv_sec=0, tv_nsec=999999593}) = -1 ETIMEDOUT (Connection timed out)
futex(0x4d2db0, FUTEX_WAKE_PRIVATE, 1)  = 1
kill(666, SIGTERM) # there we go!
...
```

Gotcha! Seems like `collector2` is one bad process. Still we need to check the last one, `collector3`:

```
➜ root@web-srv1:~$ strace -p 3613
restart_syscall(<... resuming interrupted read ...>) = -1 ETIMEDOUT (Connection timed out)
futex(0x4d5bb0, FUTEX_WAKE_PRIVATE, 1)  = 1
futex(0xc0000324c8, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x4d8460, FUTEX_WAIT_PRIVATE, 0, {tv_sec=0, tv_nsec=999999731}) = -1 ETIMEDOUT (Connection timed out)
futex(0x4d5bb0, FUTEX_WAKE_PRIVATE, 1)  = 1
futex(0xc0000324c8, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x4d8460, FUTEX_WAIT_PRIVATE, 0, {tv_sec=0, tv_nsec=999999722}) = -1 ETIMEDOUT (Connection timed out)
futex(0x4d5bb0, FUTEX_WAKE_PRIVATE, 1)  = 1
futex(0xc0000324c8, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x4d8460, FUTEX_WAIT_PRIVATE, 0, {tv_sec=0, tv_nsec=999999635}) = -1 ETIMEDOUT (Connection timed out)
futex(0x4d5bb0, FUTEX_WAKE_PRIVATE, 1)  = 1
futex(0xc0000324c8, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x4d8460, FUTEX_WAIT_PRIVATE, 0, {tv_sec=0, tv_nsec=999999641}) = -1 ETIMEDOUT (Connection timed out)
futex(0x4d5bb0, FUTEX_WAKE_PRIVATE, 1)  = 1
futex(0xc0000324c8, FUTEX_WAKE_PRIVATE, 1) = 1
...
```

Seems like only `collector2` should be terminated. First we run `ps` again to see the binary path:

```
→ root@web-srv1:~$ ps aux | grep collector2
root        3612  0.0  0.0 101916   612 ?        Sl   13:23   0:00 /bin/collector2

→ root@web-srv1:~$ kill 3613 # your PID will be different!

→ root@web-srv1:~$ ps aux | grep collector2

→ root@web-srv1:~$ rm /bin/collector2
```

Question 14 | Output redirection

Task weight: 3%

On server `app-srv1` there is a program `/bin/output-generator` which, who would've guessed, generates some output. It'll always generate the very same output for every run:

1. Run it and redirect all `stdout` into `/var/output-generator/1.out`
2. Run it and redirect all `stderr` into `/var/output-generator/2.out`
3. Run it and redirect all `stdout` and `stderr` into `/var/output-generator/3.out`
4. Run it and write the exit code number into `/var/output-generator/4.out`

Answer:

We go ahead and check the program

```
→ ssh app-srv1

→ root@app-srv1:~$ output-generator
...
2021/07/20 08:58:20 32668 - a99947a1-2f68-475e-9326-32dbd4876f60
2021/07/20 08:58:20 32669 - a99947a1-2f68-475e-9326-32dbd4876f60
2021/07/20 08:58:20 32670 - a99947a1-2f68-475e-9326-32dbd4876f60
2021/07/20 08:58:20 32671 - a99947a1-2f68-475e-9326-32dbd4876f60
2021/07/20 08:58:20 32672 - a99947a1-2f68-475e-9326-32dbd4876f60
2021/07/20 08:58:20 32673 - a99947a1-2f68-475e-9326-32dbd4876f60
2021/07/20 08:58:20 32674 - a99947a1-2f68-475e-9326-32dbd4876f60
2021/07/20 08:58:20 32675 - a99947a1-2f68-475e-9326-32dbd4876f60
2021/07/20 08:58:20 32676 - a99947a1-2f68-475e-9326-32dbd4876f60
2021/07/20 08:58:20 32677 - a99947a1-2f68-475e-9326-32dbd4876f60
```

Investigation

What a mess! Let's try to count the rows:

```
→ root@app-srv1:~$ output-generator | wc -l
...
2021/07/20 09:00:44 22670 - a99947a1-2f68-475e-9326-32dbd4876f60
2021/07/20 09:00:44 22671 - a99947a1-2f68-475e-9326-32dbd4876f60
2021/07/20 09:00:44 22672 - a99947a1-2f68-475e-9326-32dbd4876f60
2021/07/20 09:00:44 22673 - a99947a1-2f68-475e-9326-32dbd4876f60
2021/07/20 09:00:44 22674 - a99947a1-2f68-475e-9326-32dbd4876f60
2021/07/20 09:00:44 22675 - a99947a1-2f68-475e-9326-32dbd4876f60
2021/07/20 09:00:44 22676 - a99947a1-2f68-475e-9326-32dbd4876f60
20342
```

It looks like `wc -l` does only count rows that are written to stdout.

We can investigate a bit further using `stdout` (1) and `stderr` (2) redirection

```
→ output-generator > /dev/null # no stdout rows

→ output-generator 1> /dev/null # no stdout rows (same command as above)

→ output-generator 1> /dev/null 2> /dev/null # no stdout or stderr

→ output-generator 2>&1 # stderr will be redirected into stdout
```

Using this we can count all lines:

```
→ root@app-srv1:~$ output-generator 2>&1 | wc -l
32678
```

Solution

We redirect as required:

```
→ root@app-srv1:~$ output-generator > /var/output-generator/1.out

→ root@app-srv1:~$ output-generator 2> /var/output-generator/2.out

→ root@app-srv1:~$ output-generator >> /var/output-generator/3.out 2>> /var/output-generator/3.out

→ root@app-srv1:~$ output-generator

→ root@app-srv1:~$ echo $? > /var/output-generator/4.out # get the exit code and redirect to file
```

This should give us:

```
→ root@app-srv1:~$ cat /var/output-generator/1.out | wc -l
20342

→ root@app-srv1:~$ cat /var/output-generator/2.out | wc -l
12336

→ root@app-srv1:~$ cat /var/output-generator/3.out | wc -l
32678

→ root@app-srv1:~$ cat /var/output-generator/4.out
123
```

Question 15 | Build and install from source

Task weight: 4%

Install the text based terminal browser `links2` from source on server `app-srv1`. The source is provided at `/tools/links-2.14.tar.bz2` on that server.

Configure the installation process so that:

1. The target location of the installed binary will be `/usr/bin/links`
2. Support for ipv6 will be disabled

Answer:

Let's first check out the provided archive and extract it's content:

```
→ ssh app-srv1

→ root@app-srv1:~$ cd /tools

→ root@app-srv1:/tools$ ls
links-2.14.tar.bz2
```

We'll go ahead and extract the archive:

```
→ root@app-srv1:/tools$ tar xjf links-2.14.tar.bz2

→ root@app-srv1:/tools$ cd links-2.14

→ root@app-srv1:/tools/links-2.14$ ls -lh
total 6.9M
-rw-r--r-- 1 ubuntu ubuntu 9.2K Aug 12  2015 AUTHORS
-rw-r--r-- 1 ubuntu ubuntu 1.2K Jan  2  2005 BRAILLE_HOWTO
-rw-r--r-- 1 ubuntu ubuntu  19K Sep 21  2013 COPYING
-rw-r--r-- 1 ubuntu ubuntu 114K Nov 26  2016 ChangeLog
-rw-r--r-- 1 ubuntu ubuntu 4.6K Nov 26  2016 INSTALL
-rw-r--r-- 1 ubuntu ubuntu 1.9K Apr  9  2012 KEYS
-rw-r--r-- 1 ubuntu ubuntu  650 Jan 10  2005 Links_logo.png
-rw-r--r-- 1 ubuntu ubuntu 2.6K Jun 16  2016 Makefile.am
-rw-r--r-- 1 ubuntu ubuntu  22K Nov  3  2016 Makefile.in
-rw-r--r-- 1 ubuntu ubuntu  214 Feb 11  2012 NEWS
...
```

The usual process of installing from source is:

1. `./configure (args...)`
2. `make`
3. `make install`

Here the question requires specific configuration parameters. We can list all possible options:

```
➜ root@app-srv1:/tools/links-2.14$ ./configure -help
Usage: configure [options] [host]
Options: [defaults in brackets after descriptions]
Configuration:
  --cache-file=FILE       cache test results in FILE
  --help                  print this message
  --no-create             do not create output files
  --quiet, --silent       do not print checking... messages
  --version               print the version of autoconf that created configure
Directory and file names:
  --prefix=PREFIX         install architecture-independent files in PREFIX      # we need this
                          [/usr/local]
  --exec-prefix=EPREFIX   install architecture-dependent files in EPREFIX
                          [same as prefix]
  --bindir=DIR            user executables in DIR [EPREFIX/bin]
  --sbindir=DIR           system admin executables in DIR [EPREFIX/sbin]
  --libexecdir=DIR        program executables in DIR [EPREFIX/libexec]
  --datadir=DIR           read-only architecture-independent data in DIR
                          [PREFIX/share]
  --sysconfdir=DIR        read-only single-machine data in DIR [PREFIX/etc]
  --sharedstatedir=DIR    modifiable architecture-independent data in DIR
                          [PREFIX/com]
...
  --enable-graphics       use graphics
  --disable-utf8          disable UTF-8 terminal (saves memory)
  --without-getaddrinfo   compile without getaddrinfo function
  --without-ipv6          compile without ipv6                                  # we need this
  --without-libevent      compile without libevent
  --without-gpm           compile without gpm mouse
  --with-ssl(=directory)  enable SSL support
  --with-ssl=nss          enable SSL support through NSS OpenSSL emulation
  --disable-ssl-pkgconfig don't use pkgconfig when searching for openssl
...
```

It's also possible to open the `./configure` file in a text editor to investigate possible options if the help output won't suffice.

We now use the following configuration:

```
➜ root@app-srv1:/tools/links-2.14$ ./configure --prefix /usr --without-ipv6
creating cache ./config.cache
checking for a BSD compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking whether make sets ${MAKE}... yes
checking for working aclocal-1.4... missing
checking for working autoconf... missing
...
creating ./config.status
creating Makefile
creating config.h
------------------------------------------------------------
Configuration results:

Event handler:          NO
IPv6:                   NO
Supported compression:  NO
SSL support:            NO
UTF-8 terminal:         YES
GPM support:            NO
Graphics enabled:       NO
------------------------------------------------------------
```

After the configuration we continue with `make`:

```
➜ root@app-srv1:/tools/links-2.14$ make
gcc -DHAVE_CONFIG_H -I. -I. -I.     -g -O2 -c af_unix.c
gcc -DHAVE_CONFIG_H -I. -I. -I.     -g -O2 -c auth.c
gcc -DHAVE_CONFIG_H -I. -I. -I.     -g -O2 -c beos.c
gcc -DHAVE_CONFIG_H -I. -I. -I.     -g -O2 -c bfu.c
gcc -DHAVE_CONFIG_H -I. -I. -I.     -g -O2 -c block.c
gcc -DHAVE_CONFIG_H -I. -I. -I.     -g -O2 -c bookmark.c
gcc -DHAVE_CONFIG_H -I. -I. -I.     -g -O2 -c cache.c
...
gcc -DHAVE_CONFIG_H -I. -I. -I.     -g -O2 -c vms.c
gcc -DHAVE_CONFIG_H -I. -I. -I.     -g -O2 -c x.c
gcc -DHAVE_CONFIG_H -I. -I. -I.     -g -O2 -c xbm.c
gcc  -g -O2  -o links  af_unix.o auth.o beos.o bfu.o block.o bookmark.o cache.o charsets.o compress.o connect.o
cookies.o data.o default.o dip.o directfb.o dither.o dns.o dos.o drivers.o error.o file.o finger.o fn_impl.o
font_inc.o framebuf.o ftp.o gif.o grx.o hpux.o html.o html_gr.o html_r.o html_tbl.o http.o https.o img.o imgcache.o
jpeg.o jsint.o kbd.o language.o listedit.o lru.o mailto.o main.o memory.o menu.o objreq.o os_dep.o pmshell.o png.o
sched.o select.o session.o smb.o string.o suffix.o svg.o svgalib.o terminal.o tiff.o types.o url.o view.o view_gr.o
vms.o x.o xbm.o  -lpthread
```

```
/usr/bin/ld: session.o: in function `get_temp_name':
/tools/links-2.14/session.c:1056: warning: the use of `tempnam' is dangerous, better use `mkstemp'
```

Followed by the `make install`:

```
➜ root@app-srv1:/tools/links-2.14$ make install
make[1]: Entering directory '/tools/links-2.14'
/bin/sh ./mkinstalldirs /usr/bin
  /usr/bin/install -c  links /usr/bin/links
make  install-man1
make[2]: Entering directory '/tools/links-2.14'
/bin/sh ./mkinstalldirs /usr/man/man1
 /usr/bin/install -c -m 644 ./links.1 /usr/man/man1/links.1
make[2]: Leaving directory '/tools/links-2.14'
make[1]: Leaving directory '/tools/links-2.14'
```

This should result in:

```
➜ root@app-srv1:~$ whereis links
links: /usr/bin/links /usr/man/man1/links.1

➜ root@app-srv1:~$ /usr/bin/links -version
Links 2.14
```

We can verify that ipv6 is disabled:

```
➜ root@app-srv1:~$ /usr/bin/links -lookup ip6-localhost
error: host not found          # output if ipv6 is disabled

➜ root@app-srv1:~$ /usr/bin/links -lookup ip6-localhost
::1                            # output if ipv6 is enabled
```

Question 16 | LoadBalancer

Task weight: 8%

Server `web-srv1` is hosting two applications, one accessible on port `1111` and one on `2222`. These are served using Nginx and it's **not** allowed to change their config. The ip of `web-srv1` is `192.168.10.60`.

Create a new HTTP LoadBalancer on that server which:

- Listens on port `8001` and redirects all traffic to `192.168.10.60:2222/special`
- Listens on port `8000` and balances traffic between `192.168.10.60:1111` and `192.168.10.60:2222` in a Random or Round Robin fashion

Nginx is already preinstalled and is recommended to be used for the implementation. Though it's also possible to use any other technologies (like Apache or HAProxy) because only the end result will be verified.

Answer:

First we check if everything works as claimed:

```
➜ curl web-srv1:1111
app1

➜ curl web-srv1:2222
app2

➜ curl web-srv1:2222/special
app2 special
```

Cool, we can work with that! Here we're now going to create an Nginx LoadBalancer, but it would also be possible to use any other technology if you're more familiar with it.

```
➜ student@terminal:~$ ssh web-srv1

➜ root@web-srv1:~$ cd /etc/nginx/sites-available

➜ root@web-srv1:/etc/nginx/sites-available$ ls -lh
total 2.0K
-rw-r--r-- 1 root root 329 Jun 14 17:17 app1
-rw-r--r-- 1 root root 413 Jun 14 17:17 app2

➜ root@web-srv1:/etc/nginx/sites-available$ cat app1
server {
        listen 1111 default_server;
```

```
        listen [::]:1111 default_server;

        server_name _;

        location / {
                return 200 'app1\n';
        }
}
```

There we see the two existing applications `app1` and `app2` which we aren't allowed to be changed. But we sure can use one as template for our new LoadBalancer:

```
➜ root@web-srv1:/etc/nginx/sites-available$ cp app1 lb

➜ root@web-srv1:/etc/nginx/sites-available$ vim lb
```

```
server {
        listen 8001 default_server;                         # change port
        listen [::]:8001 default_server;                    # change port

        server_name _;

        location / {
                proxy_pass http://192.168.10.60:2222/special; # reverse proxy to the requsted url
        }
}
```

We start slowly with the easier one on port `8001` where we simply use a `proxy_pass` option. Save the file and:

```
➜ root@web-srv1:/etc/nginx/sites-available$ cd ../sites-enabled

➜ root@web-srv1:/etc/nginx/sites-enabled$ ln -s ../sites-available/lb . # this is the nginx way for enabling

➜ root@web-srv1:/etc/nginx/sites-enabled$ service nginx restart

➜ root@web-srv1:/etc/nginx/sites-enabled$ curl localhost:8001          # our LB is reachable on port 8001
app2 special
```

Working! Now we go ahead and copy the first part, make some additions and use it as the second part:

```
➜ root@web-srv1:/etc/nginx/sites-enabled$ vim lb
```

```
### FIRST PART
server {
        listen 8001 default_server;
        listen [::]:8001 default_server;

        server_name _;

        location / {
                proxy_pass http://192.168.10.60:2222/special; # reverse proxy to the requsted url
        }
}

### SECOND PART
upstream backend {
        # when no load balancing method is specified, Round Robin is used
        server 192.168.10.60:1111; # app1
        server 192.168.10.60:2222; # app2
}

server {
        listen 8000 default_server;
        listen [::]:8000 default_server;

        server_name _;

        location / {
                proxy_pass http://backend; # reverse proxy to the requsted url
        }
}
```

The second part is mostly the same as before. It's possible to create multiple servers which listen on different ports within the same file. Here we simply created an `upstream backend` that contains the provided urls and we use it in the `proxy_pass` directive. Pretty nice right! But does it work?

```
➜ root@web-srv1:/etc/nginx/sites-enabled$ service nginx restart

➜ root@web-srv1:/etc/nginx/sites-enabled$ exit

➜ curl web-srv1:8000
app1

➜ curl web-srv1:8000
```

```
app2

➜ curl web-srv1:8000
app1

➜ curl web-srv1:8000
app2

➜ curl web-srv1:8000
app1

➜ curl web-srv1:8001
app2 special

➜ curl web-srv1:8001
app2 special

➜ curl web-srv1:8001/anything/even/not/special
app2 special
```

Above we see that requests to `web-srv1:8000` are sent to both `app1` and `app2`. And requests on `web-srv1:8001` are only sent to `app2` and path `/special`.

Question 17 | OpenSSH Configuration

Task weight: 6%

You need to perform OpenSSH server configuration changes on `data-002`. Users `marta` and `cilla` exist on that server and can be used for testing. Passwords are their username and shouldn't be changed. Please go ahead and:

1. Disable `X11Forwarding`
2. Disable `PasswordAuthentication` for everyone but user `marta`
3. Enable `Banner` with file `/etc/ssh/sshd-banner` for users `marta` and `cilla`

> ***NOTE:*** In case of misconfiguration you can still access the instance using `sudo lxc exec data-002 bash`

Answer:

Step 1

We are required to perform ssh server config changes, always fun because nothing can ever go wrong!

We're doing a simple one first:

```
➜ ssh data-002

➜ root@data-002:~$ vim /etc/ssh/sshd_config
```

```
...
X11Forwarding no  # find and update to "no"
...
```

Save the file and restart the ssh service:

```
➜ root@data-002:~$ service ssh restart # no error output means good
```

Step 2+3

We now need to first disable `PasswordAuthentication` globally and then enable it for user `marta`. Then we also add the `Banner` settings for users `marta` and `cilla`:

```
➜ root@data-002:~$ vim /etc/ssh/sshd_config
```

```
...
PasswordAuthentication no  # find and update to "no"
...
# go to very end of the file and add:
Match User marta
  PasswordAuthentication yes
  Banner /etc/ssh/sshd-banner

Match User cilla
  Banner /etc/ssh/sshd-banner
```

It's **very important** to add any `Match` lines at the very bottom of the config file, otherwise it might not get accepted and errors will be thrown during sshd service restart.

Using `Match User` or `Match Group` we can override global settings for specific users and groups. Let's test if it works:

```
➜ root@data-002:~$ service ssh restart

➜ root@data-002:~$ exit

➜ ssh marta@data-002
Hello our favorite user!                                # user "marta" sees banner message
marta@data-002's password:
Last login: Sat Jun 17 16:13:35 2023 from 192.168.10.1  # user "marta" can log in using password

➜ marta@data-002:~$ exit

➜ ssh cilla@data-002
Hello our favorite user!                                # user "cilla" sees banner message
cilla@data-002: Permission denied (publickey).          # user "cilla" can't log in using password

➜ ssh root@data-002
Last login: Sat Jun 17 16:21:30 2023 from 192.168.10.1  # no banner message for user "root"
```

Both users `marta` and `cilla` see the banner message, but only `marta` can still log in using password.

Question 18 | LVM Storage

Task weight: 8%

You're required to perform changes on LVM volumes:

1. Reduce the volume group `vol1` by removing disk `/dev/vdh` from it
2. Create a new volume group named `vol2` which uses disk `/dev/vdh`
3. Create a `50M` logical volume named `p1` for volume group `vol2`
4. Format that new logical volume with `ext4`

Answer:

Some helpful abbreviations when working with LVM, because command names usually start with those:

```
PV = Physical Volume
VG = Volume Group
LV = Logical Volume
```

Start by having a look at PVs:

```
➜ sudo pvs
  PV         VG        Fmt  Attr PSize    PFree
  /dev/vda3  ubuntu-vg lvm2 a--  <126.00g 63.00g
  /dev/vdg   vol1      lvm2 a--    96.00m 84.00m
  /dev/vdh   vol1      lvm2 a--    96.00m 96.00m
```

In the output above we can see that the VG `vol1` uses two disks `/dev/vdg` and `/dev/vdh`. We can also get an overview over all system disks and their LVM usage:

```
➜ sudo lvmdiskscan
  /dev/loop0 [     <49.84 MiB]
  /dev/loop1 [      63.28 MiB]
  /dev/loop2 [    <111.95 MiB]
  /dev/vda2  [       2.00 GiB]
  /dev/loop3 [     <53.26 MiB]
  /dev/vda3  [    <126.00 GiB] LVM physical volume
  /dev/loop4 [      63.45 MiB]
  /dev/vdb   [     100.00 MiB]
  /dev/vdc   [     100.00 MiB]
```

```
  /dev/vdd   [     100.00 MiB]
  /dev/vde   [     100.00 MiB]
  /dev/vdf   [     100.00 MiB]
  /dev/vdg   [     100.00 MiB] LVM physical volume # disk 1
  /dev/vdh   [     100.00 MiB] LVM physical volume # disk 2
  5 disks
  6 partitions
  2 LVM physical volume whole disks
  1 LVM physical volume
```

The existing PV `/dev/vda3` with VG `ubuntu-vg` is created by the main operating system and shouldn't be touched.

Step 1

We want to remove disk `/dev/vdh` from the existing VG `vol1`:

```
→ sudo vgs # list all VGs
  VG         PV  LV  SN Attr   VSize    VFree
  ubuntu-vg   1   1   0 wz--n- <126.00g  63.00g
  vol1        2   1   0 wz--n-  192.00m 180.00m # two PVs

→ sudo vgreduce vol1 /dev/vdh
  Removed "/dev/vdh" from volume group "vol1"

→ sudo vgs
  VG         PV  LV  SN Attr   VSize    VFree
  ubuntu-vg   1   1   0 wz--n- <126.00g 63.00g
  vol1        1   1   0 wz--n-   96.00m 84.00m  # one PV
```

That should do it. We can also verify this by listing all PVs:

```
→ sudo pvs
  PV         VG        Fmt  Attr PSize    PFree
  /dev/vda3  ubuntu-vg lvm2 a--  <126.00g  63.00g
  /dev/vdg   vol1      lvm2 a--    96.00m  84.00m
  /dev/vdh             lvm2 ---   100.00m 100.00m # not assigned to a VG any longer
```

Step 2

Now we're going to create a new `PV` using that now free disk:

```
→ sudo vgcreate vol2 /dev/vdh
  Volume group "vol2" successfully created

→ sudo pvs
  PV         VG        Fmt  Attr PSize    PFree
  /dev/vda3  ubuntu-vg lvm2 a--  <126.00g 63.00g
  /dev/vdg   vol1      lvm2 a--    96.00m 84.00m
  /dev/vdh   vol2      lvm2 a--    96.00m 96.00m # assigned to the VG
```

Step 3

We continue by creating a LV for our new VG:

```
→ sudo lvs # no LV yet for vol2
  LV        VG        Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  ubuntu-lv ubuntu-vg -wi-ao---- <63.00g
  p1        vol1      -wi-a-----  12.00m

→ sudo lvcreate --size 50M --name p1 vol2
  Rounding up size to full physical extent 52.00 MiB
  Logical volume "p1" created.

→ sudo lvs
  LV        VG        Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  ubuntu-lv ubuntu-vg -wi-ao---- <63.00g
  p1        vol1      -wi-a-----  12.00m
  p1        vol2      -wi-a-----  52.00m  # there we go
```

Step 4

We can access LVM partitions or LVs in the usual way once we know the path:

```
➜ sudo mkfs -t ext4 /dev/vol2/p1
mke2fs 1.46.5 (30-Dec-2021)
Creating filesystem with 13312 4k blocks and 13312 inodes

Allocating group tables: done
Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting information: done
```

We could now go ahead and mount and use `/dev/vol2/p1` as we're used to.

Extending a LV

Also interesting and could also be part of the exam is extending a mounted LV:

```
➜ sudo mkdir /mnt/vol2_p1

➜ sudo mount /dev/vol2/p1 /mnt/vol2_p1

➜ sudo fdisk -l
...
Disk /dev/mapper/vol2-p1: 52 MiB, 54525952 bytes, 106496 sectors # ~50M
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes

➜ sudo lvs
  LV        VG        Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  ubuntu-lv ubuntu-vg -wi-ao---- <63.00g
  p1        vol1      -wi-a-----  12.00m
  p1        vol2      -wi-ao----  52.00m

➜ sudo lvresize vol2/p1 --size 70M # raise to 70M
  Rounding size to boundary between physical extents: 72.00 MiB.
  Size of logical volume vol2/p1 changed from 52.00 MiB (13 extents) to 72.00 MiB (18 extents).
  Logical volume vol2/p1 successfully resized.

➜ sudo fdisk -l
...
Disk /dev/mapper/vol2-p1: 72 MiB, 75497472 bytes, 147456 sectors # ~70M
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

Question 19 | Regex, filter out log lines

Task weight: 4%

On server `web-srv1` there are two log files that need to be worked with:

1. File `/var/log-collector/003/nginx.log`: extract all log lines where URLs start with `/app/user` and that were accessed by browser identity `hacker-bot/1.2`. Write only those lines into `/var/log-collector/003/nginx.log.extracted`
2. File `/var/log-collector/003/server.log`: replace all lines starting with `container.web`, ending with `24h` and that have the word `Running` anywhere in-between with: `SENSITIVE LINE REMOVED`

Answer:

First we find the files in the specified location

```
➜ ssh web-srv1

➜ root@web-srv1:~$ cd /var/log-collector/003

➜ root@web-srv1:/var/log-collector/003$ ls -lha
total 23K
drwxr-xr-x 2 root root    4 Jul 19 09:43 .
drwxr-xr-x 8 root root    8 Jul 19 09:43 ..
-rw-r--r-- 1 root root 207K Jul 19 09:40 nginx.log
-rw-r--r-- 1 root root  29K Jul 19 09:39 server.log
```

Step 1

To extract all log lines as required we could try some simple `grep` like:

```
→ root@web-srv1:/var/log-collector/003$ cat nginx.log | grep "/app/user" | grep "hacker-bot/1.2"
```

But this would also catch lines like these which are not asked for:

```
127.0.0.1 - - [18/Jul/2075:08:35:34 +0000] "GET /hacker-bot/1.2 HTTP/1.1" 200 8 "-" "/app/user" "-"
127.0.0.1 - - [18/Jul/2075:08:35:15 +0000] "GET /hacker-bot/1.2 HTTP/1.1" 200 8 "-" "/app/user" "-"
```

The lines above shouldn't match because the url is `hacker-bot/1.2` and **NOT** the browser identity.

So we better use a simple regex

```
→ root@web-srv1:/var/log-collector/003$ cat nginx.log | grep -E "/app/user.*hacker-bot/1.2"
```

It should be 27 lines:

```
→ root@web-srv1:/var/log-collector/003$ cat nginx.log | grep -E "/app/user.*hacker-bot/1.2" | wc -l

27
```

So we write it to the required location:

```
→ root@web-srv1:/var/log-collector/003$ cat nginx.log | grep -E "/app/user.*hacker-bot/1.2" > nginx.log.extracted
```

Step 2

Next we shall remove some sensitive logs in `server.log`. Anything in the pattern of:

```
container.web ... Running ... 24h
```

In regex this could be:

```
^container.web.*Running.*24h$
```

Let's give this a go:

```
→ root@web-srv1:/var/log-collector/003$ cat server.log | grep -E "^container.web.*Running.*24h$"
```

It should be 44 lines:

```
→ root@web-srv1:/var/log-collector/003$ cat server.log | grep -E "^container.web.*Running.*24h$"  | wc -l

44
```

To replace these we can use `sed` using the same regex

```
→ root@web-srv1:/var/log-collector/003$ sed 's/^container.web.*Running.*24h$/SENSITIVE LINE REMOVED/g' server.log
```

This will simply output everything to stdout for us to verify. We can even further check by counting the lines:

```
→ root@web-srv1:/var/log-collector/003$ sed 's/^container.web.*Running.*24h$/SENSITIVE LINE REMOVED/g' server.log |
grep "SENSITIVE LINE REMOVED" | wc -l

44
```

Looks fine, 44 lines again. Now we can use `sed` to replace the actual file. Still, always make a backup!

```
→ root@web-srv1:/var/log-collector/003$ cp server.log server.log.bak # backups ftw

→ root@web-srv1:/var/log-collector/003$ sed -i 's/^container.web.*Running.*24h$/SENSITIVE LINE REMOVED/g' server.log

→ root@web-srv1:/var/log-collector/003$ cat /var/log-collector/003/server.log | grep 'SENSITIVE LINE REMOVED' | wc -l
44
```

The 44 we see above is the result we want.

Question 20 | User and Group limits

Task weight: 6%

User Jackie caused an issue with her account on server `web-srv1`. She did run a program which created too many subprocesses for the server to handle. A coworker of yours already solved it temporarily and limited the number of processes user `jackie` can run.

For this the coworker added a command into `.bashrc` in the home directory of `jackie`. But the command just sets the soft limit and not the hard limit. Jackies password is `brown` in case needed.

Configure the number-of-processes limitation as a hard limit for user `jackie`. Use the same number currently set as a soft limit for that user. Do it in the proper way, not via `.bashrc`.

While at it you remember another ticket. On the same server you should enforce that group `operators` can only ever log in once at the same time.

Answer:

Step 1

Well, that's a lot. We check out that user and its limits first:

```
➜ ssh web-srv1

➜ root@web-srv1:~$ su jackie

➜ jackie@web-srv1:/root$ cd

➜ jackie@web-srv1:~$

➜ jackie@web-srv1:~$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 7815
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 1024 # that's he one!
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

Using `ulimit` we can see all configured limits, and there is `max user processes` set to 1024. We should check how it has been set in `.bashrc`:

```
➜ jackie@web-srv1:~$ if ! shopt -oq posix; then
...
ulimit -S -u 1024 # current implementation
```

This works, but user `jackie` could change it herself like this:

```
➜ jackie@web-srv1:~$ ulimit -u # list current
1024

➜ jackie@web-srv1:~$ ulimit -S -u 1100

➜ jackie@web-srv1:~$ ulimit -u
1100
```

First we remove that line that was added as a temporary fix

```
➜ root@web-srv1:~$ vim .bashrc # remove the ulimit setting
```

Next we go ahead and configure it via `/etc/security/limits.conf` (we need to be root for this). In that file there are already some useful examples on the bottom. We add a new line

```
➜ root@web-srv1:~$ vim /etc/security/limits.conf
```

```
...
jackie          hard    nproc           1024     # add a new line
```

Now we can confirm our change:

```
➜ jackie@web-srv1:~$ ulimit -u
1024

➜ jackie@web-srv1:~$ ulimit -S -u 1100
bash: ulimit: max user processes: cannot modify limit: Invalid argument
```

User `jackie` is not able any longer to change that limit herself.

Step 2

The other ticket was about implementing a limitation for group `operators`, so we add it to the file as well:

```
→ root@web-srv1:~$ vim /etc/security/limits.conf
```

```
...
jackie          hard    nproc           1024
@operators      hard    maxlogins       1       # add this line
```

We can test it using user `jackie` because she actually is in group `operators`. For this we can to ssh into `web-srv1` with `jackie:brown`:

```
→ ssh jackie@web-srv1
jackie@web-srv1's password: brown

→ jackie@web-srv1:~$ su jackie
Password: brown
Too many logins for 'jackie'.
su: cannot open session: Permission denied
```

kim wüstkamp