Part 1: Contenidors LXC

Hem estat usant els contenidors LXC però sense entrar en detalls. En aquesta sessió baixarem a baix nivell.

1. Configuració

- Vegeu el contingut dels fitxers /etc/lxc/*
- Feu:

```
lxc-config -l
```

Els seu sistema de fitxers és accessibles des del host amfitrió i es troben a: **/var/lib/lxc/router/rootfs/**:

```
lxc-config lxc.lxcpath
/home/milax/.local/share/lxc
sudo lxc-config lxc.lxcpath
/var/lib/lxc/
```

- Vegeu **man lxc** per a veure la estructura.

→ En lloc d'usar el sistema de fitxers de l'amfitrió es poden usar volums lògics (LVM) d'una certa mida (vegeu man lxc-create --bdev)

✓ Els fitxers són **persistents** entre execucions.

A aquest lxc.path també hi ha el fitxer **config** on es defineix la configuració de cada contenidor.

- Vegeu el contingut pels contenidors router i server.

2. Cgroups

- Feu **man lxc.container.conf**
  → busqueu-hi l'exemple de cgroups COMPLEX CONFIGURATION

Alguns (pocs) paràmetres es poden modificar en calent:

```
lxc-cgroup -n router cpuset.cpus "1"
```

3. Namespaces

Per a veure des de l'amfitrió els *namespaces*[3] es pot fer amb la comanda *ps*:

```
ps -h -o pid,pidns,netns,cmd
```

3 *feu: man network_namespaces ip-netns*

```
...
  89065 4026532571 4026532573 /sbin/agetty -o -p -- \u --noclear --keep-baud
console 115200,38400,9600 vt220
  89148 4026532506 4026532508 /sbin/agetty -o -p -- \u --noclear --keep-baud
console 115200,38400,9600 vt220
  89849 4026532240 4026532242 /sbin/agetty -o -p -- \u --noclear --keep-baud
console 115200,38400,9600 vt220
  89892 4026532441 4026532443 /sbin/agetty -o -p -- \u --noclear --keep-baud
console 115200,38400,9600 vt220
...
```

Per a veure els processos d'un contenidor:

```
      pgrep -a --ns 89892
      88659 /sbin/init
      88747 /lib/systemd/systemd-journald
      89892 /sbin/agetty -o -p -- \u --noclear --keep-baud console
115200,38400,9600 vt220
      93418 /bin/bash
      93756 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
      93796 /usr/sbin/dhcpd -4 -q -cf /etc/dhcp/dhcpd.conf eth0
```

Exemple de network namespace

Crearem en calent una nova interfície pel contenidor server i la posem al pont de la intranet (i la aixequem per dhcp):

```
PID=$(lxc-info -pH server)
ip link add name veth0_host type veth peer name veth2_server
ip link set veth0_host master lxcbr2
ip link set dev veth2_server netns $PID name eth2
ip link set dev veth0_host up
```

Comproveu les noves interfícies:

```
      ip -c link
      brctl show lxcbr2
```

Aixequeu la nova interfície del sever:

```
      lxc-attach server
      dhclient -4 eth2
```

Part 2: Contenidors Docker

Per a crear contenidors **docker**[4] podem treballar amb contenidors personalitzats (pel que fa al paquets instal·lats i la seva configuració) o usar imatges disponibles a Internet (de docker hub o d'altres Container Registries).

Per a poder treballar amb docker sense privilegis elevats podem afegir els usuaris al grup *docker.* Si l'usuari milax no pertany al grup cal afegir-li:

```
sudo usermod -aG docker milax
newgrp docker # canviar el grup actual
```

En aquest darrer cas si feu 'exit' llavors tronareu al grup anterior.

El docker als linux gestiona la seva pròpia xarxa virtual, usant el bridge **docker0** i manipulant les regles del tallafocs (iptables).

Pas 1. Crear imatge base

Fem una instal·lació mínima a partir de docker-hub i provem les comandes bàsiques.

```
docker pull debian:bullseye-slim
docker images
REPOSITORY  TAG            IMAGE ID       CREATED         SIZE
debian      bullseye-slim  500fb7ca53f5   30 hours ago    80.6MB
```

Les imatges es construeixen a partir d'altres formant un **overlay** i no són tan accessibles com als contenidors LXC:

```
sudo ls -la /var/lib/docker/image/overlay2/imagedb/content/sha256 | grep --
color  500fb7ca53f5
```

Per a més informació (de la imatge, en particular dels *overlays*) proveu:

```
docker image inspect 500fb7ca53f5 | grep overlay2
```

4 *Si a la distro no hi és proveu amb el paquet* docker.io *o bé amb* docker-ce

Pas 2. Execució d'un contenidor en primer pla.

Executem i comprovem que no podem fer quasi res.

```
docker run -ti --name prova debian:bullseye-slim
root@796f53fc5b39:/# ip address
bash: ip: command not found
root@796f53fc5b39:/# hostname -I
172.17.0.2
root@796f53fc5b39:/# exit
docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
# no apareix el contenidor, però hi és:
docker container ls -a
CONTAINER ID   IMAGE                  COMMAND   CREATED        STATUS
PORTS     NAMES
94e658d70c88   debian:bullseye-slim   "bash"    3 minutes ago  Exited (0)  prova
docker container rm prova
```

No hi ha editors, ni systemd, network-manager, iptables, ps, sysctl, cron ... Llavors, com molts paquets no estan instal·lats, haurem de recórrer a obtenir informació al /proc (*process information pseudo-filesystem*):

Per veure la taula d'encaminament, les IPs i si fa el *forwarding* de datagrames:

```
cat /proc/net/route
cat /proc/net/fib_trie
cat /proc/sys/net/ipv4/ip_forward
```

> Cal dir que els contenidor docker són efímers. Proveu d'executar-ne un, instal·lar un paquet, sortir i, en tornar a fer el run el paquet ja no hi és[5].

Una alternativa seria comprovar del de l'amfitrió amb les seves comandes. Per això primer necessitem saber el PID del contenidor. Per a obtenir-lo proveu en un altre terminal:

```
docker top prova
docker inspect --format='{{.State.Pid}}' prova
```

També podem entrar al *network namespace* del contenidor i comprovem l'estat de la xarxa:

```
sudo nsenter --target $PID --net ip -c address
sudo nsenter -t $PID -n ip route
```

5 *El mateix passa amb les configuracions que poseu, per exemple, a /etc/**

Pas 3. Crear una imatge millor

Podem crear una imatge a mida nostra, incorporant els paquets i els fitxers necessaris, usant les eines **docker-build** o docker-compose, sent el primer el més senzill. Necessitarem un fitxer Dockerfile. Per exemple:

```
# Fitxer: dockerfile_gsx_prac1
# vim: syntax=dockerfile

FROM debian:bullseye-slim
MAINTAINER Professor GSX

RUN apt update
RUN apt-get install -y --no-install-recommends \
      iproute2 bind9-host iputils-ping ifupdown

COPY ./prac1_gsx.tgz /root

WORKDIR     /root
RUN   tar -xzf prac1_gsx.tgz
```

Per a construir la imatge:

```
docker build -t gsx:prac1 -f dockerfile_gsx_prac1 .
docker images
REPOSITORY      TAG             IMAGE ID       CREATED         SIZE
gsx             prac1           544522123e3a   7 seconds ago   147MB
debian          bullseye-slim   500fb7ca53f5   31 hours ago    80.6MB
```

Engeguem un contenidor i comprovem que malgrat ser root no podem fer ho tot doncs no tenim certes 'capacitats':

```
docker run -ti --rm --network=ISP --hostname router --name Router gsx:prac1
root@router:~# ip address flush dev eth0
Failed to send flush request: Operation not permitted
```

Per a evitar-ho haurem d'executar els contenidors amb més *capabilities.* Afegim el paràmetre: **--cap-add=NET_ADMIN**

Pas 4: pràctica 3 amb contenidors docker

Creeu el fitxer dockerfile_gsx_prac3[6] fent que s'instal·lin els següent paquets: dnsutils ifupdown iproute2 iptables iputils-ping isc-dhcp-client nano procps rsyslog tcpdump

```
docker build -t gsx:prac3 -f dockerfile_gsx_prac3 .
```

Per a reproduir un entorn "similar" al de la pràctica 3 haurem de crear les xarxes i els contenidors pertinents:

```
docker network create --driver=bridge --subnet=10.0.2.16/30 ISP
docker network create --driver=bridge --subnet=$IPSDMZ \
--gateway=$IP2aDMZ DMZ
docker network create --driver=bridge --subnet=$IPSINTRA \
--gateway=$IP2aINTRA INTRANET

docker network ls
docker network inspect DMZ
```

Observació: quan creem una docker network (bridge) la primera IP la assigna al bridge, el qual farà de *gateway*. A nosaltres no ens convé (perquè és la que usem pel router). Per això especifiquen que la IP del *gateway* sigui la segona (en principi no usada) i no importarà perquè a la prac3 ja es configura el *gateway* de cada contenidor.

Per a executar el contenidor router ho fem amb:

```
OPCIONS="-itd --rm --cap-add=NET_ADMIN"
# -t, --tty=true|false
# -i, --interactive=true|false
# -d, --detach=true|false
# --rm true|false Automatically remove the container when it exits.
imatge="gsx:prac3"
docker run $OPCIONS --hostname router --network=ISP --name Router $imatge
docker network connect DMZ Router
docker network connect INTRANET Router
xterm -e docker attach Router &
```

Observació: al docker run sols es pot especificar una xarxa. Per això l'executem 'detached' i després fem les altres connexions. A la resta de contenidors això no caldrà.

6 *Si voleu podeu fer-la a partir de la imatge gsx:prac1*

Pot ser que necessitéssim més *capabilities*:

```
root@router:~/router# dmesg
dmesg: read kernel buffer failed: Operation not permitted
```

Caldrà afegir el paràmetre --cap-add=**SYS_ADMIN** i tornar a fer el run.

Segurament haureu de fer algun retoc as scripts per a adaptar-se a les limitacions dels paquets instal·lats als contenidors. En particular no tindrem el servei ssh i al no tenir el systemd no podrem usar systemctl, journalctl, etc.

Si modifiqueu els fitxers per a no perdre'ls els podeu enviar per scp des dels contenidors cap a un servidor ssh exterm (per exemple a milax@casa).

L'objectiu és trastejar amb el docker, no cal que us capfiqueu en que funcioni tot.

Referències bàsiques:

- man docker, docker-image, docker-container, docker-container-run, etc
- man capabilities