Les conteneurs Linux

Un conteneur Linux est un terme générique pour une implémentation d'une forme de virtualisation au niveau du système d'exploitation. Il s'agit en effet d'une manière d'isoler un ou plusieurs processus qui tournent en leur créant un environnement qui sera différent, parfois à l'extrême, des autres processus tournant sur le même système.

À la différence de la virtualisation classique, une seule copie du système d'exploitation tourne sur la machine, les conteneurs peuvent ainsi réclamer moins de ressources en termes de mémoire.

A propos de conteneurs

Les conteneurs LXC

Le terme Linux Container (LXC) désigne à la fois les applications virtualisées basées sur Linux. A la différence des machines virtuelles qui fonctionnent au niveau du matériel, les conteneurs fonctionnent au niveau du système d'exploitation, Les conteneurs se partagent le système d'exploitation et isolent les processus d'application du reste du système tandis que les systèmes de virtualisation classique permettent l'exécution de plusieurs systèmes d'exploitation sur un seul système.

Les conteneurs Linux nécessitent en général moins de ressources qu'une machine virtuelle et possèdent une interface standard qui permet de gérer facilement plusieurs conteneurs simultanément.

Un inconvénient majeur de LXC se situe au niveau de la gestion de la mémoire : bien que différents backend de mémoire (ivm, overlayfs, zfs et btrfs) soient pris en charge, par défaut, la mémoire est stockée directement sur le rootfs.

Le RootFS

Un "rootfs" est "le système de fichiers que la partition racine" (le /). La structure d'un RootFS peut servir de base à des conteneurs:

1. sur une machine, lorsque l'installation normale ne peut pas fonctionner (ex. hardware trop récent pour le kernel d'installation)
2. LXC
3. OpenVZ
4. VServer
5. UML
6. coLinux

Préparation d'un RootFS

RootFS prêts à l'emploi

linuxcontainers.org proposent des RootFS en téléchargement gratuit sur:

1. > https://jenkins.linuxcontainers.org/
2. > https://images.linuxcontainers.org/

Les distributiopns proposent également des rootfs:

1. > https://github.com/oracle/container-images

Les image disques

Par exemple Les images pour Raspberry Pi3 et Pi4 disponibles depuis https://raspi.debian.net/, sont utilisées pour graver une carte SD directement amorçable. On peut donc extraire un RootFS depuis ces images.

Télécharger l'erchive:

```
wget
https://raspi.debian.net/tested/20231111_raspi_4_trixie.img.{xz,xz.sha256.asc}
```

Les images pour Raspberry Pi3 et Pi4 sont mises à disposition par Gunnar Wolf / gwolf, mainteneur Debian. Rechercher l'empreinte de sa clé à partir https://nm.debian.org/public/people/ puis https://nm.debian.org/person/gwolf/

```
gpg --keyserver keyring.debian.org --recv-keys
4D14050653A402D73687049D2404C9546E145360
```

Vérifier la signature des checksums et la checksum des fichiers images compressés

```
gpg --verify 20231111_raspi_4_trixie.img.xz.sha256.asc
sha256sum -c 20231111_raspi_4_trixie.img.xz.sha256.asc
```

Extraire l'image et les partitions:

```
unxz 20231111_raspi_4_trixie.img.xz
7z x 20231111_raspi_4_trixie.img
```

La première partition est le boot et contient le noyau et les firms, elle peut être extraite avec 7z. La seconde est le FS ext4 et doit être montée pour en récupérer le contenu:

```
7z x 0.fat
fuse2fs -o ro 1.img /mnt
```

```
cp -a /mnt/* /tmp/rootfs/
umount /mnt
```

On peut récupérer des RootFS directement sur le site https://downloads.raspberrypi.com/

Les conteneurs LXC

Il est possible d'amorcer un RootFS avec lxc-create:

```
lxc-create -t download -n debian11 -- -d debian -r bullseye -a arm64
echo "root:$(echo 'raspberry' | openssl passwd -6 -stdin)" | sed -i -e
"s|^root:[^:]\+:|$(awk '{print $1}'):|"
/var/lib/lxc/debian11/rootfs/etc/shadow
sed -i 's/^\[Resolve\]/[Resolve]\nDNS=8.8.8.8 8.8.4.4\nFallbackDNS=1.1.1.1
1.0.0.1/g' /var/lib/lxc/debian11/rootfs/etc/systemd/resolved.conf
```

Les images turnkey

Le site mirrors.dotsrc.org propose des images turnkey qui peuvent être utilisées dans les conteneurs lxc

Cependant les services **inithooks** déployés avec le package **inithooks** exécute des scripts d'initialisation du système qui régénèrent les clés secrètes (par exemple, SSH, certificat SSL par défaut), définit des mots de passe (par exemple, root, base de données, application) et configure les paramètres d'application de base (par exemple, domaine, e-mail d'administrateur). Ces services doivent être désactivés afin de gérer manuellement ces configurations.

```
mkdir -pv /tmp/{iso,rootfs}
wget
https://mirrors.dotsrc.org/turnkeylinux/images/proxmox/debian-11-turnkey-jen
kins_17.1-1_amd64.tar.gz -P /tmp/iso
tar xf /tmp/iso/debian-11-turnkey-jenkins_17.1-1_amd64.tar.gz -C
/tmp/rootfs/
cd /var/lib/lxc/jenkins/rootfs
rsync -aPHxv /tmp/rootfs/etc/systemd/system/multi-user.target.wants/
/var/lib/lxc/jenkins/rootfs/etc/systemd/system/multi-user.target.wants/
rm /tmp/rootfs/etc/systemd/system/multi-user.target.wants/inithooks.service
rm /tmp/rootfs/etc/systemd/system/basic.target.wants/inithooks-*.service
rsync -aPHxv /tmp/rootfs/etc/ssh /var/lib/lxc/jenkins/rootfs/etc/ssh/
echo "root:$(echo 'raspberry' | openssl passwd -6 -stdin)" | sed -i -e
"s|^root:[^:]\+:|$(awk '{print $1}'):|" /tmp/rootfs/etc/shadow
cp -a /lib/systemd/system/ssh.service /tmp/rootfs/lib/systemd/system/
echo "PermitRootLogin yes" >> /tmp/rootfs/etc/ssh/sshd_config
echo "net.ipv6.conf.all.disable_ipv6 = 1" >> /tmp/rootfs/etc/sysctl.d/99-
```

```
sysctl.conf
```

Virtualisation avec qemu

À la différence de la virtualisation classique, une seule copie du système d'exploitation tourne sur la machine, les conteneurs peuvent ainsi réclamer moins de ressources en termes de mémoire.

Pour émuler un autre noyau Linux sur lequel un conteneur poura fonctionner, QEMU peut être utilisé pour l'exécuter. Cela permet d'isoler les applications et de les exécuter de manière plus légère que les machines virtuelles traditionnelles.

Configuration du système hôte

Il convient d'installer l'émulateur qemu ainsi que libvirt:

```
sudo apt install -y git  qemu-system qemu-system-arm qemu-system-aarch64
qemu-utils lxc bridge-utils lxc lxc-templates libvirt0 libpam-cgfs bridge-
utils uidmap unzip cpio
sudo apt install -y libvirt-clients libvirt-daemon-system virtinst virt
```

Pour pouvoir démarrer sur un RootFS, un serveur NFS est également requis :

```
sudo apt install -y nfs-kernel-servel
```

Lorsque le système hôte est lui même émulé dans un namespace (un conteneur), il faut utiliser un serveur NFS qui s'exécute en user namespace, par exemple **unfs3**:

- Télécharger le paquet `https://cloudfront.debian.net/debian-archive/debian/pool/main/u/unfs3/unfs3_0.9.21+dfsg-1_amd64.deb`
- Installer le paquet manuellement `dpkg -i unfs3_0.9.21+dfsg-1_amd64.deb`
- Télécharger le scripte unfsd.init `wget https://raw.githubusercontent.com/unfs3/unfs3/refs/heads/master/unfsd.init -P /etc/init.d' * Modifier la commande start /usr/sbin/unfsd -u -n 2049 -m 2049 -i ${pidfile}'`
- Editer le nfs-user-server.service pour définir l'unité systemd

```
[Unit]
Description=NFS server and services
DefaultDependencies=no
Requires= network.target

[Service]
Type=oneshot
```

```
RemainAfterExit=yes
ExecStart=/etc/init.d/unfsd start
ExecStop=/etc/init.d/unfsd stop
ExecReload=/usr/sbin/exportfs -r

[Install]
WantedBy=multi-user.target
```

virsh net-define default.xml

Chargement du rootfs dans une archive portable cpio

Lorsque le rootfs n'est pas trop gros, on peut le charger directement en mémoire avec l'option **-initrd** et fournir un fichier cpio (compressé) avec le rootfs. Donc, si on a un fichier tar, il faut le décompresser et créer un cpio à la place.

```
wget
http://cdimage.debian.org/mirror/turnkeylinux/images/iso/turnkey-jenkins-17.
1-bullseye-amd64.iso
isoinfo -R -X turnkey-jenkins-17.1-bullseye-amd64.iso
7z x turnkey-jenkins-17.1-bullseye-amd64.iso
7z x live/initrd.gz
sqfs2tar /tmp/iso/live/10root.squashfs > /tmp/iso/rootfs.tar
cd /tmp/rootfs
cpio -idv < /tmp/iso/mkinitramfs-MAIN_HXrvlm
echo "root:$(echo 'raspberry' | openssl passwd -6 -stdin)" | sed -i -e
"s|^root:[^:]\+:|$(awk '{print $1}'):|" etc/shadow
sed -i 's/^\[Resolve\]/[Resolve]\nDNS=8.8.8.8 8.8.4.4\nFallbackDNS=1.1.1.1
1.0.0.1/g' etc/systemd/resolved.conf
tar -uvf /tmp/iso/rootfs.tar .
```

Il faut utiliser le format '-H newc' pour cpio.

```
cd /tmp/iso
find . | cpio --quiet -o -H newc | bzip2 -c > /qemu-rpi-kernel/rootfs.cpio
```

Il est nécessaire ,au minimum, de définir un utilisateur admin ou de définir un mot de passe pour l'utilisateur root (non recomandé) pour pouvoir se connecter:

```
echo "root:$(echo 'raspberry' | openssl passwd -6 -stdin)" | sed -
i -e "s|^root:[^:]\+:|$(awk '{print $1}'):|" etc/shadow
```

ou

```
echo 'password' |openssl passwd -1 -salt QidasQKZ -stdin | cut -d“
” -f1 | xargs -I{} sed -i
'/^root/s/\([^:]*\):[^:]*:\(.*\)/\1:'"{}"':\2/'
/tmp/rootfs/etc/shadow
```

Il est important de noter que la propriété et les autorisations de tous les fichiers du système de fichiers racine doivent être préservées. Pour cela, il faut exécuter tar en tant que root (via sudo) et utilisez l'argument –same-owner dans la commande.

La décomprsesion de cerains items spéciaux (/dev) peuvent ressortir en erreur, il n'est pas nécessaire de décompresser toute l'archive tar afin de la covertir au format cpio. **bsdtar** permet de convertir un fichier tar au format cpio:

```
cd /tmp/iso
bsdtar --format=newc -cf @rootfs.tar > /qemu-rpi-
kernel/rootfs.cpio
```

Pour modifier un fichier spécifique il suffit de l'extraire:

```
tar -xf rootfs.tar.xz "/etc/shadow"
```

Puis de mettre à jour l'archive avec le fichier modifié:

```
tar -uvf rootfs.tar etc/shadow
```

rsync -aPHxv /tmp/rootfs/ /var/lib/lxc/jenkins/rootfs/

boot nfsroot

Il est possible de passer les options au noyau linux pour démarrer directement depuis un export NFS '-append “ip=dhcp root=/dev/nfs rw nfsroot=10.0.3.1:/tmp/rootfs/,vers=3”`.

Mais pour cela les options de démarrage des stacks IP sont requises (l'option « Racine du système de fichiers sur NFS » est également requise), car toutes les autres machines configurent le réseau dans leurs scripts de démarrage. La compilationdu noyau avec les options suivantes permet la configuration automatique des adresses IP des périphériques et de la table de routage lors du démarrage du noyau:

```
CONFIG_IP_PNP -- IP: kernel-level configuration support
CONFIG_IP_PNP_ENABLE
```

```
CONFIG_IP_PNP_DHCP -- DHCP support
CONFIG_NFS_FS -- NFS files ystem support
CONFIG_NFS_V3 -- NFS version 3 support
CONFIG_ROOT_NFS -- Root file system on NFS
```

initrd avec nfsroot

Si le noyau utilisé ne dispose pas des options de démarrage des stacks IP requises on peut utiliser un initrd busybox pour démarrer en NFS.

Le processus est le suivant:

1. démarrage sur un initrd minimal (busybox)
2. montage du rootfs en nfs
3. switch_root[1)] vers le point de montage

```
ARCH=arm64
URL="http://deb.debian.org/debian/pool/main/b/busybox/busybox-static_1.37.0-4_${ARCH}.deb"
package=$(basename "$URL")
wget -nv -nc "$URL"
rm -rf "initramfs.${ARCH}.cpio" "initramfs.${ARCH}.cpio.gz"
dpkg-deb -x "$package" tmp/
mkdir -pv tmp/etc/udhcpc
cat > tmp/etc/udhcpc/default.script << 'EOL'
#!/bin/sh

# cargo cult from buildroot
[ -z "$1" ] && echo "Error: should be called from udhcpc" && exit 1
case "$1" in
  renew|bound)
  /sbin/ifconfig $interface $ip $BROADCAST $NETMASK
  if [ -n "$router" ] ; then
    while route del default gw 0.0.0.0 dev $interface 2>/dev/null; do
      :
    done
    for i in $router ; do
      route add default gw $i dev $interface
    done
  fi
  ;;
esac

RESOLV_CONF="/etc/resolv.conf"

echo -n > $RESOLV_CONF
[ -n "$domain" ] && echo domain $domain >> $RESOLV_CONF
for i in $dns
do
```

```
  echo adding dns $i
  echo nameserver $i >> $RESOLV_CONF
done
EOL

cat > tmp/init << 'EOL'
#!/bin/busybox sh
echo "Loading, please wait..."

mkdir -p /newroot /proc /sbin /sys /tmp /usr/bin /usr/sbin /var/lock
/bin/busybox --install
mount -t sysfs -o nodev,noexec,nosuid none /sys
mount -t proc -o nodev,noexec,nosuid none /proc

depmod -a||true

# Modprobe all drivers/net/ethernet modules
for module in $(find /lib/modules/$(uname -r)/kernel/drivers/net/ethernet -
type f -name '*.ko') ; do
        module_name=$(basename "${module}")
        modprobe ${module_name%.ko}||true
done

modprobe bridge||true

echo  "Bringing up bridge ..."
ifconfig eth0 0.0.0.0 up
brctl addbr br0
brctl addif br0 eth0
ifconfig br0 0.0.0.0 up
udhcpc -i br0 -q

echo "Sleep 5 ..."
sleep 5

# parsing stolen from initramfs-tools

for x in $(cat /proc/cmdline); do
        case "$x" in
        rootdelay=*)
                ROOTDELAY="${x#rootdelay=}"
                case ${ROOTDELAY} in
                *[![:digit:].]*)
                        ROOTDELAY=
                        ;;
                esac
                ;;
        nfsroot=*)
                NFSROOT="${x#nfsroot=}"
                ;;
        esac
```

```
done

# get nfs root from dhcp
if [ "x${NFSROOT}" = "xauto" ]; then
        # check if server ip is part of dhcp root-path
        if [ "${ROOTPATH#*:}" = "${ROOTPATH}" ]; then
                NFSROOT="${ROOTSERVER}:${ROOTPATH}"
        else
                NFSROOT="${ROOTPATH}"
        fi

# nfsroot=[<server-ip>:]<root-dir>[,<nfs-options>]
elif [ -n "${NFSROOT}" ]; then
        # nfs options are an optional arg
        if [ "${NFSROOT#*,}" != "${NFSROOT}" ]; then
                NFSOPTS="-o ${NFSROOT#*,}"
        fi
        NFSROOT=${NFSROOT%%,*}
        if [ "${NFSROOT#*:}" = "$NFSROOT" ]; then
                NFSROOT="${ROOTSERVER}:${NFSROOT}"
        fi
fi

echo "Mount NFS ..."
mount -t nfs -o nolock ${NFSOPTS} ${NFSROOT} /newroot||/bin/busybox sh -i
</dev/console >/dev/console 2>&1

ln -sf /run/systemd/resolve/resolv.conf /newroot/etc/resolv.conf
# populate interfaces to keep NM and ifupdown away
cat > /newroot/etc/network/interfaces <<EOF
iface eth0 inet static
iface br0 inet dhcp
EOF

10.0.3.1:/tmp/rootfs /newroot/ nfs rw,bg,soft,intr,nosuid 0 0
mount --move /proc /newroot/proc
mount --move /sys /newroot/sys

echo "Switch to NFS rootfs"
exec switch_root /newroot /sbin/init
EOL

cd tmp
find . | cpio -H newc -o > "../initramfs.${ARCH}.cpio"
gzip -9 "initramfs.${ARCH}.cpio"
rm -rf tmp/
```

Le groupe apertis propose des kits complets (noyau + initrd) permettant d'amorcer un RootFS en NFS. Ces kits sont disponibles dans les dossiers nfs des release sur le site https://images.apertis.org/

Exéxution de l'image

Depuis une image cpio

```
qemu-system-x86_64 -machine virt -m 4G -kernel /qemu-rpi-kernel/Image-5.16.1
-initrd /qemu-rpi-kernel/etherpad.cpio -device virtio-net-pci,netdev=user0 -
netdev bridge,id=user0,br=virbr0  -nographic
```

Cette méthode charge l'image du rootfs en mémoire RAM, donc tous les changements sont volatiles et disparaitront quand la machine sera arrêtée.

Pour garder les modifications il faut mettre à jour le rootfs source par exemple avec un rsync, puis recréer l'image cpio.

```
rsync -aPHxv  -e 'ssh -o StrictHostKeyChecking=no' --exclude-
from=/root/exclude-files.txt  root@192.168.122.76:/
/var/lib/lxc/debian11/rootfs/
```

Il faut impérativement exclure de la synchronization les dossiers proc,tmp,sys,boot et dev:

```
vi /root/exclude-files.txt
/boot
/dev
/tmp
/sys
/proc
/backup
/etc/mtab
/etc/mdadm.conf
```

Architecture native (x86)

- En mode non graphique avec sortie directe sur le sysout par défaut (terminal)

```
qemu-system-aarch64 -machine virt -cpu cortex-a72 -smp 6 -m 2G -kernel
/qemu-rpi-kernel/Image-5.16.1 -initrd /qemu-rpi-kernel/rootfs.cpio -device
virtio-net-pci,netdev=user0 -netdev bridge,id=user0,br=virbr0  -nographic -
append "ip=dhcp rdinit=/sbin/init root=/dev/ram rootfstype=ramfs rw nokaslr"
```

- En mode daemon sans sortie sur le sysout

```
qemu-system-aarch64 -machine virt -cpu cortex-a72 -smp 6 -m 2G -kernel
/qemu-rpi-kernel/Image-5.16.1 -initrd /qemu-rpi-kernel/rootfs.cpio -device
virtio-net-pci,netdev=user0 -netdev bridge,id=user0,br=virbr0  -display none
-daemonize -append "ip=dhcp rdinit=/sbin/init root=/dev/ram rootfstype=ramfs
```

```
rw nokaslr"
```

- depuis un livecd

```
qemu-system-x86_64 -cdrom turnkey-jenkins-17.1-bullseye-amd64.iso -m 2G -
device virtio-net-pci,netdev=user0 -netdev bridge,id=user0,br=virbr0  -
display none
```

- avec un dossier du system-host comme rootfs

```
qemu-system-x86_64 -machine virt -m 2G -kernel /tmp/rootfs/vmlinuz -initrd
/tmp/rootfs/initrd.img -device virtio-net-pci,netdev=user0 -netdev
bridge,id=user0,br=virbr0 -nographic -append "ip=dhcp rdinit=/sbin/init
root=/dev/nfs rw nfsroot=192.168.122.1:/tmp/rootfs/"
```

- avec un kernel spécifique

```
qemu-system-x86_64 -smp 6 -m 4G -kernel /qemu-rpi-kernel/x86_64-image-6.1.1
-initrd /qemu-rpi-kernel/jenkins.cpio -device virtio-net-pci,netdev=user0 -
netdev bridge,id=user0,br=virbr0 -nographic -curses -append "ip=dhcp
rdinit=/sbin/init root=/dev/nfs rw
nfsroot=192.168.122.1:/tmp/rootfs/,tcp,vers=3"
```

Foreign architecture

Les sites suivants proposent des kernels pour les architectures arm:

1. >https://images.apertis.org/
2. > https://github.com/dhruvvyas90/qemu-rpi-kernel.git
3. > https://l4re.org/download/Linux-kernel/ (recommandé car le kernel est compilé avec les options IP et NFS)

On suppose qemu installé pour l'architecture arm 64 bits (aarch64)

```
qemu-system-aarch64 --version
QEMU emulator version 5.0.0 (Debian 1:5.0-14~bpo10+1)
Copyright (c) 2003-2020 Fabrice Bellard and the QEMU Project developers
```

Vérifier la présence du modèle virt:

```
qemu-system-aarch64 -machine help |grep virt
....
virt                 QEMU 5.0 ARM Virtual Machine (alias of virt-5.0)
virt-5.0             QEMU 5.0 ARM Virtual Machin
```

On peut utiliser le noyau téléchargé afin d'émuler une architecture arm64 dans un système x86:

```
qemu-system-aarch64 -machine virt -cpu cortex-a53 -smp 6 -m 2G -kernel
/qemu-rpi-kernel/vmlinuz-5.7.0-qcomlt-arm64 -initrd /qemu-rpi-
kernel/initrd-5.0.4-img -device virtio-net-pci,netdev=user0 -netdev
bridge,id=user0,br=lxcbr0  -nographic -curses -append "ip=dhcp
rdinit=/sbin/init root=/dev/ram rootfstype=ramfs rw nokaslr"
```

```
qemu-system-aarch64 -machine virt -cpu cortex-a72 -smp 6 -m 2G -kernel
/qemu-rpi-kernel/l4re/kernel/arm64/Image-6.6.8 -initrd /qemu-rpi-
kernel/initramfs.arm64.cpio -device virtio-net-pci,netdev=user0 -netdev
bridge,id=user0,br=lxcbr0 -nographic -append "earlycon rootdelay=5
console=ttyAMA0 ip=dhcp root=/dev/nfs rw nfsroot=10.0.3.1:/tmp/rootfs/"
```

```
qemu-system-aarch64 -machine virt -cpu cortex-a72 -smp 2 -m 4G -kernel
/qemu-rpi-kernel/l4re/kernel/arm64/Image-6.6.8 -device virtio-net-
pci,netdev=user0 -netdev bridge,id=user0,br=lxcbr0 -nographic -append
"earlycon rootdelay=5 console=ttyAMA0 ip=dhcp root=/dev/nfs rw
nfsroot=10.0.3.1:/tmp/rootfs/,vers=3"
```

1)

la commande **switch_root** permet de passer à un autre système de fichiers racine de l'arbre de montage