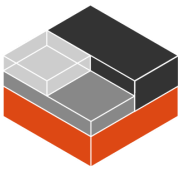
TD Linux Container (LXC)
VIRTUALISATION LINUX

version 2021
ENSEIGNANT M. ORNECH
jean-francois.ornech@veniseverteniort.org

Pré-requis: Ce TD doit être réalisé sur une machine Debian de préférence. Vous devrez posséder un accès root pour certaines commandes. Si ce n'est pas le cas, utilisez une machine virtuelle.

Introduction

Qu'est-ce que LXC ?

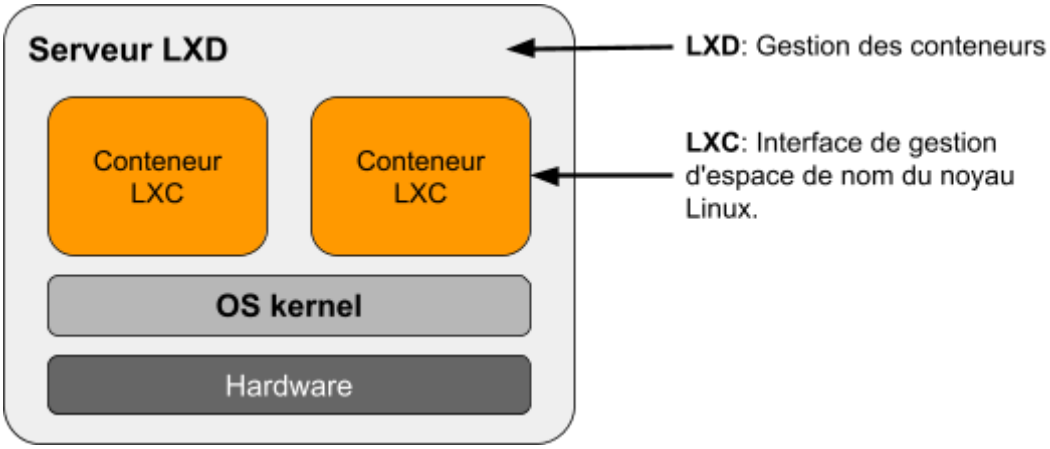


LXC est un ensemble d'outils et de librairies propre au système Linux dédiée à l'exécution de systèmes Linux isolés (les conteneurs). Lxc s'appuie sur les namespaces et les cgroups pour faire fonctionner ces containers. On distingue deux types de containers : fat (gras) et thin (mince). Cette terminologie est empruntée aux jails BSD.

- Un thin jail revient à containeriser une application.
- Un fat jail consiste à faire tourner un système entier dans un container.

Qu'est-ce que LXD ?

LXD est un outil de gestion des containers LXC. Il offre des fonctionnalités comme la mise en cluster, la sauvegarde, les snapshots, etc..



Comment est-ce que cela fonctionne ?

Les conteneurs LXC (*LinuX Containers*) s'appuient sur deux technologies du noyau Linux, les groupes de contrôle (cgroups) et les espaces de nom (namespaces).

1. Les **cgroups** (groupes de contrôle) permettent d'allouer, de limiter et de prioriser l'accès aux ressources (Cpu, mémoire, accès disque, bande passante réseau, etc ...) .
2. Les **namespaces** (espace de nom) permettent une isolation complète d'une application dans l'environnement système comme l'arborescence des processus, le réseau, les utilisateurs et le montage des systèmes de fichiers.

Autre élément important, les conteneurs LXC (*LinuX Containers*) partagent le même noyau que la machine hôte. Les containers hébergés doivent donc être compatibles avec le noyau de la machine hôte .

TD1: Les groupes de contrôle (cgroup)

Un groupe de contrôle est une fonctionnalité du noyau Linux qui permet la gestion hiérarchique et l'allocation des ressources système. Par exemple, il est possible de limiter l'usage de la RAM, du processeur ou de l'espace disque d'un processus.

Pour bien comprendre ce mécanisme, il nous faut revenir sur la notion de processus du point vue Unix. A l'origine, c'est-à -dire tout de suite après la phase de boot du système, un premier processus est créé (/bin/init) . Il a pour fonction de démarrer les processus nécessaires au fonctionnement du système, comme les daemons et les processus gettys (gestion d'un terminal) qui pourront à leur tour créer des processus. Il est donc question ici d'une arborescence et d'héritage entre processus où le processus /bin/init est au sommet de la hiérarchie. La première conséquence est que si vous détruisez un processus, l'ensemble des processus fils (c'est-à-dire tous ceux créés par ce dernier) seront également détruits. La seconde est que ce modèle d'arborescence des processus Unix, où le processus /bin/init est au sommet de la pyramide, se compose d'une seule et unique arborescence.

L'avancée majeure proposée par les cgroups (en comparaison au modèle hiérarchisé des processus Linux) est qu'il est maintenant possible de faire cohabiter plusieurs arborescences de processus sans qu'elles soient interconnectées par un processus parent. Avec les cgroups , toutes les arborescences de processus sont rattachées, non pas à un processus parent, mais à un ou plusieurs sous-systèmes. On entend par sous-système, une ressource unique comme un temps processeur ou un accès mémoire.

Vous trouverez la définition officielle des cgroups ici:
https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt

N'hésitez pas à vous rendre sur ce lien si la notion de sous-système ou de cgroups n'est pas claire pour vous.

Pour afficher la liste des sous-systèmes tapez la commande:

```
# ls -l /sys/fs/cgroup
total 0
dr-xr-xr-x 4 root root  0 oct.   6  2021 blkio
```

```
lrwxrwxrwx 1 root root 11 oct.   6  2021 cpu -> cpu,cpuacct
lrwxrwxrwx 1 root root 11 oct.   6  2021 cpuacct -> cpu,cpuacct
dr-xr-xr-x 4 root root  0 oct.   6  2021 cpu,cpuacct
dr-xr-xr-x 2 root root  0 oct.   6  2021 cpuset
dr-xr-xr-x 4 root root  0 oct.   6  2021 devices
dr-xr-xr-x 2 root root  0 oct.   6  2021 freezer
dr-xr-xr-x 2 root root  0 oct.   6  2021 hugetlb
dr-xr-xr-x 4 root root  0 oct.   6  2021 memory
lrwxrwxrwx 1 root root 16 oct.   6  2021 net_cls -> net_cls,net_prio
dr-xr-xr-x 2 root root  0 oct.   6  2021 net_cls,net_prio
lrwxrwxrwx 1 root root 16 oct.   6  2021 net_prio -> net_cls,net_prio
dr-xr-xr-x 2 root root  0 oct.   6  2021 perf_event
dr-xr-xr-x 4 root root  0 oct.   6  2021 pids
dr-xr-xr-x 2 root root  0 oct.   6  2021 rdma
dr-xr-xr-x 5 root root  0 oct.   6  2021 systemd
dr-xr-xr-x 5 root root  0 oct.   6  2021 unified
```

Voici quelques description des cgroups :

- **blkio (Block IO Controller):** Permet de gérer les accès aux périphériques de type block comme les disques durs.
- **cpu:** Utilise le scheduler pour fournir aux processus des cgroups
- **cpuacct (CPU Accounting Controller):** Permet de générer automatiquement des rapports sur les ressources CPU utilisées par les processus d'un cgroup.
- **cpuset:** Permet d'assigner un ou plusieurs CPU physiques aux processus d'un cgroup
- **devices (Device Whitelist Controller):** Autorise ou interdit l'accès à certains périphériques
- **freezer:** Gèle ou relance les processus d'un cgroup
- **memory (Memory Resource Controller):** Permet de limiter l'utilisation de la mémoire par les processus d'un cgroup
- **net_cls (Network classifier):** Identifier les paquets réseau en provenance d'un cgroup en y ajoutant un tag de type classid
- **net_prio (Network priority):** Permet de définir dynamiquement la priorité du trafic réseau généré par diverses applications
- **pids:** empêche toute nouvelle tâche d'être clonée après qu'une certaine limite ait été atteinte.

Installez les quelques outils :

```
# apt install libcgroup1 cgroup-tools
```

Créons dans le sous-système `memory` notre premier cgroup que nous nommerons "test". Attention ces commandes devront être exécutées depuis l'utilisateur root. La commande "sudo" ne sera pas suffisante. Utilisez une machine virtuelle Linux si ce n'est pas votre cas.

Commencez par créer un répertoire test dans /sys/fs/cgroup/memory
```
# mkdir /sys/fs/cgroup/memory/test
```

Et voilà, notre premier cgroup est créé.

Par défaut un cgroup créé dans le sous-système memory héritera de la totalité de l'espace mémoire disponible. Pour limiter l'utilisation de la mémoire à 30 Mo aux processus attachés à ce cgroup, il suffit de créer un fichier `memory.limit_in_bytes` contenant la valeur souhaitée en octet.
```
# echo 30000000 > /sys/fs/cgroup/memory/test/memory.limit_in_bytes
```

Créons un script que nous nommerons test.sh:
```
# nano test.sh
```
```
#!/bin/bash
while [1]; do
      echo "Salut tout le monde"
      sleep 60
done
```

Crtl+S pour sauvegarder et Ctrl+X pour quitter

Lancez le script test.sh
```
# sh ./test.sh &
```

Récupérez le PID du processus de votre script précédemment lancé.
```
# ps -ax | grep test.sh
   6475 pts/0    S      0:00 sh ./test.sh
   6506 pts/0    S+     0:00 grep --color=auto test.sh
```

Pour mon exemple, le PID de mon script est le 6475.

> **Rappel:**
> L'identifiant de processus ou PID (**P**rocess **ID**entifier) est une valeur unique attribuée par le système à chaque processus lors de son démarrage.

Maintenant que nous connaissons le PID de notre script, voyons de quoi a-t-il hérité en terme d'accès système.
```
# ps -ww -o cgroup 6475
CGROUP
12:blkio:/user.slice,11:memory:/user.slice/user-1000.slice/user@1000.service,7:devices:/user.slice,5:pids:/user.slice/user-1000.slice/user@1000.service,2:cpu,cpuacct:/user.slice,1:name=systemd:/user.slice/user-1000.slice/user@1000.service/apps.slice/apps-org.gnome.T
```

```
erminal.slice/vte-spawn-3b52bc88-7b0c-42af-8b44-a1ba5b420b2c.scope
,0::/user.slice/user-1000.slice/user@1000.service/apps.slice/apps-
org.gnome.Terminal.slice/vte-spawn-3b52bc88-7b0c-42af-8b44-a1ba5b4
20b2c.scope
```

On observe que notre processus est rattaché au sous-système memory:
`11:memory:/user.slice/user-1000.slice/user@1000.service,`

Cependant il a hérité des accès en lien avec l'arborescence de la connexion sh de notre utilisateur.

Pour profiter de la limitation des 30Mo de notre cgroup test nous devons deplacer le processus 6476 vers le cgroup test.

```
# echo 6475 > /sys/fs/cgroup/memory/test/cgroup.procs
# ps -ww -o cgroup 6475
```

```
11:memory:/test
```

Bien évidemment vous utiliserez votre PID, pas celui de mon exemple.
On constate cette fois-ci que le processus du script test.sh est lié au cgroup test du sous-système memory. Le reste est inchangé.

Maintenant qu'il fait partie du cgroup test, nous pouvons également le monitorer.

```
# cat /sys/fs/cgroup/memory/test/memory.usage_in_bytes
221286
```

L'occupation de la mémoire est d'environ 200ko.

Détruisez le processus de test.sh

```
# kill 6475
```

Créez un second cgroup test2

```
# mkdir /sys/fs/cgroup/memory/test2
```

Limitez les ressources mémoire à 5 Ko (4096 octets est la taille minimum d'une page mémoire).

```
# echo 5000 > /sys/fs/cgroup/memory/memory.limit_in_bytes

# cat /sys/fs/cgroup/memory/test/memory.usage_in_bytes
```

Lancez le script test.sh

```
# sh ./test.sh &
```

Récupérez son PID du processus en.

```
# ps -ax | grep test.sh
   6532 pts/0    S      0:00 sh ./test.sh
```

Déplacez le processus vers le cgroup test2

```
# echo 6532 > /sys/fs/cgroup/memory/test2/cgroup.procs
```

Attendez quelques secondes… une erreur se produit. En effet le processus ne possède pas assez d'espace mémoire pour fonctionner le système là donc éjecté .

Les espaces de nom (namespace)

Un namespace est une instance des ressources système disponible sur un hôte. Cette instance a pour fonction d'isoler un processus du contexte système.

Voici les différents espaces de nom disponibles sur un système Linux.

Namespace	fonction
cgroups	Virtualisent la vue des processus appartenant à un groupes de contrôle notamment la vue des répertoires /proc/[pid]/cgroup et /proc/[pid]/mountinfo.
PID	isole l'arborescence d'un processus
mount	isole les points de montage du système hôte
network	isole les périphériques réseaux , port, etc…
user	isole les IDs des utilisateurs et groupes du système hôte
time	permet de virtualiser l'horloge système
uts	isole les hostnames et noms de domaine

Dans les faits chaque processus possède son propre sous-répertoire /proc/[pid]/ns/ contenant une entrée pour chaque espace de nom pouvant être manipulé (notamment avec la commande `setns`)

```
$ ls -l /proc/$$/ns | awk '{print $1, $9, $10, $11}'
```

Lors de la création d'un nouvel espace de nom , l'isolation se fait sur trois axes :

1. Lorsqu'un nouvel namespace PID est initialisé, le processus prend le numéro de PID 1;
2. Une nouvelle pile réseau est allouée au processus. Dès lors aucun conflit d'adresse est possible avec les services réseaux de l'hôte même s'il possède la même IP;
3. Les systèmes de fichiers sont indépendants, on peut monter et démonter des volumes sans que cela ait une incidence sur l'hôte.

Les principales commandes pour la manipulation des espaces de nom sont:

- La commande `clone` pour la création d'un nouvel espace de nom
- La commande `setns` permet de joindre un processus à un espace de nom existant
- La commande `unshare` déplace un processus vers un espace de nom existant
- La commande `ioctl` permet de récupérer et manipuler les relations entre espaces de nom existant

Rendez-vous sur ce lien pour plus de détails sur les espaces de nom.

Pour illustrer le fonctionnement des namespaces nous allons utiliser la commande `unshare`. La commande `unshare` crée un nouveau namespace, puis exécute le programme spécifié dans ce nouveau namespace . Notez que si aucune application n'est spécifiée, le Shell par défaut sera exécuté.

```
# unshare --fork --pid --mount-proc bash
```

Voyons un peu plus près ce que nous avons fait:

1. Nous avons lancé un `bash` grâce à `unshare`
2. L'option `--fork` en a fait un fils de unshare
3. L'option `--pid` autorise la création de son propre namespace PID. Notre processus possède le PID 1.
4. L'option `--mount-proc` autorise la création d'un système de fichier proc. Pour rappel, le répertoire /proc contient une arborescence de fichiers spéciaux qui représentent l'état actuel du noyau ; permettant aux applications et aux utilisateurs d'obtenir un aperçu du système du point du vue du noyau.

```
# ps -aux
USER  PID  %CPU  %MEM     VSZ  RSS     TTY  STAT  START  TIME  COMMAND
root    1   0.0   0.1  115440  2024  pts/0     S  21:32  0:00  bash
root   10   0.0   0.1  155360  1872  pts/0    R+  21:32  0:00  ps -aux
```

Noter que ce n'est plus le processus /sbin/init qui possède le PID 1 mais notre `bash` lancé depuis la commande `unshare`. `bash` est bien détaché de l'arborescence des processus de notre hôte grâce à son nouvel espace de nom. Il sera donc le père de tous les processus lancés à partir de ce namespace.

Pour plus de détails sur la commande `unshare`, tapez `man unshare` ou rendez-vous sur ce lien.

Sortez de cet espace de nom

```
# exit
```

Affiche de nouveau les processus
```
# ps -aux
```

Nous retrouvons bien l'arborescence des processus de notre hôte, avec le processus /sbin/init possédant le PID numéro 1.

TP2: Les containers Linux LXC

Installation

La grande majorité des fonctionnalités étant déjà présentes dans le noyau de la machine hôte, l'installation des paquets nécessaires au fonctionnement d'un container ne prendra pas plus que quelques Mo.
```
# apt update
# apt upgrade
# apt install lxc lxc-templates
```

Pour information, les templates seront installés dans le répertoire `/usr/share/lxc/templates` et leurs configurations respectives dans `/usr/share/lxc/config`.

Pour installer `lxd`, utilisez le gestionnaire de packages snap:
```
# apt install snapd
# snap install core
# snap install lxd
```

Vérifiez votre configuration avec la commande:
```
# lxc-checkconfig
```

Configuration initiale de LXD

Avant son utilisation Lxd doit être initialisé. Pour cela procédez comme ceci:
```
# lxd init
```
- Would you like to use LXD clustering? (yes/no) [default=no]: **no**
- Do you want to configure a new storage pool? (yes/no) [default=yes]: **yes**
- Name of the new storage pool [default=default]: **default**
- Name of the storage backend to use (btrfs, dir, lvm, zfs, ceph) [default=zfs]: **dir**
- Would you like to connect to a MAAS server? (yes/no) [default=no]: **no**
- Would you like to create a new local network bridge? (yes/no) [default=yes]: **yes**
- What should the new bridge be called? [default=lxdbr0]: **lxdbr0**
- What IPv4 address should be used? [default=auto]: **auto**
- What IPv6 address should be used? [default=auto]: **none**
- Would you like the LXD server to be available over the network? (yes/no) [default=no]: **yes**
- Address to bind LXD to (not including port) [default=all]: **all**

- Port to bind LXD to [default=8443]: 8843
- Would you like the LXD server to be available over the network? (yes/no) [default=no]: **yes**
- Address to bind LXD to (not including port) [default=all]: **all**
- Trust password for new clients: <mot de passe>
- Again:<mot de passe>
- Would you like stale cached images to be updated automatically? (yes/no) [default=yes]: **yes**
- Would you like a YAML "lxd init" preseed to be printed? (yes/no) [default=no]: **no**

Affichez la liste des repositories distants d'où vous pouvez télécharger les images.

```
# lxc remote list
```

```
+-----------------+-----------------------------------------+---------------+-------------+--------+--------+--------+
|      NAME       |                  URL                    |   PROTOCOL    |  AUTH TYPE  | PUBLIC | STATIC | GLOBAL |
+-----------------+-----------------------------------------+---------------+-------------+--------+--------+--------+
| images          | https://images.linuxcontainers.org      | simplestreams | none        | YES    | NO     | NO     |
+-----------------+-----------------------------------------+---------------+-------------+--------+--------+--------+
| local (current) | unix://                                 | lxd           | file access | NO     | YES    | NO     |
+-----------------+-----------------------------------------+---------------+-------------+--------+--------+--------+
| ubuntu          | https://cloud-images.ubuntu.com/releases | simplestreams | none        | YES    | YES    | NO     |
+-----------------+-----------------------------------------+---------------+-------------+--------+--------+--------+
| ubuntu-daily    | https://cloud-images.ubuntu.com/daily   | simplestreams | none        | YES    | YES    | NO     |
+-----------------+-----------------------------------------+---------------+-------------+--------+--------+--------+
```

Pour afficher la liste des images disponibles, utilisez cette commande:

```
# lxc image list images: | more
```

N'omettez pas les : à la fin du nom du repository ubuntu. Vous obtiendrez une liste vide.

La colonne "alias" désigne le nom de l'image que vous devrez utiliser pour installer cette image.

Test

Création du premier conteneur

lxc launch ubuntu:18.04 conteneur1

Premier conteneur

LXC permet aussi bien de créer de petits containers contenant le minimum vital que de gros containers intégrant une distribution Linux complète . Pour ce premier exemple, nous allons utiliser le template lxc-busybox. Busybox est un container ne pesant pas plus de 2 Mo .

Pour créer notre premier container, taper la commande:

```
# lxc-create -n premier_container -t busybox
```

L'option -n va permettre de nommer notre container et l'option -t de spécifier le template que l'on souhaite utiliser.

Une fois créé nous pouvons démarrer notre containers avec la commande :

```
# lxc-start premier_container
```

Il ne nous reste plus qu'à se connecter en mode console à ce conteneur avec la commande :

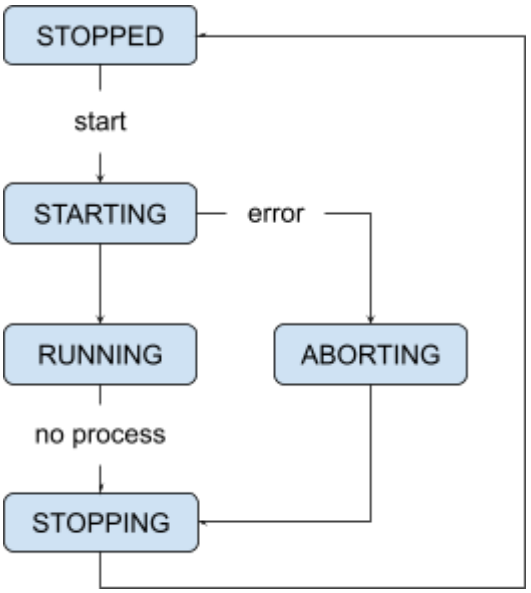
```
# lxc-attach premier_container
```

Notre conteneur est immédiatement fonctionnel.

Pour sortir du mode console du containers taper la commande `exit` ou bien faite la combinaison de touche Ctrl+D.
Noter que notre conteneur continue de tourner malgré qu'on soit sorti du mode console. On peut le vérifier avec la commande:

```
# lxc-ls -f --running
```

Cycle de vie d'un container



Lorsqu'un conteneur est lancé, une première phase d'initialisation est exécutée . Le conteneur passe d'abord par un l'état STARTING. Si cette phase se déroule correctement, il changera d'état passera alors en RUNNING.

En cas de défaillance lors de l'initialisation, il passera dans l' état ABORTING, puis sera stoppé.

Notez que si le dernier processus exécuté à l'intérieur du conteneur se ferme, le conteneur se stoppé. Il changera d'état automatiquement et passera en mode STOPPING, puis STOPPED.

Pour visualiser l'état de notre premier_container, utilisez la commande `lxc-info`:

```
# lxc-info premier_container
Name:           premier_container
State:          RUNNING
PID:            31972
CPU use:        0.01 seconds
BlkIO use:      8.00 KiB
Memory use:     1.13 MiB
KMem use:       936.00 KiB
```

Maintenant stoppons ce containers :

```
# lxc-stop premier_container
```

Affichez les détails de notre conteneur

```
# lxc-info premier_container
Name:           premier_container
State:          STOPPED
```

Création d'un container ubuntu

Créons un container système à partir de l'image ubuntu:18.04 que nous nommerons ubuntu.

```
sudo lxc launch ubuntu:18.04 ubuntu
```

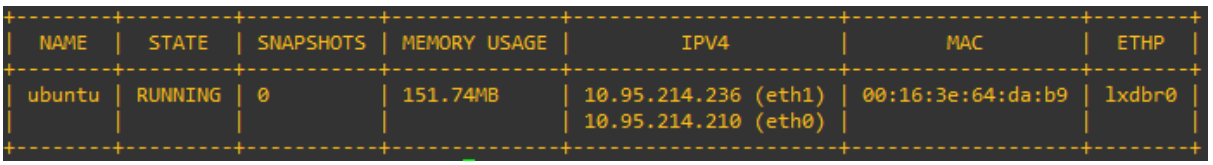
Comme précédemment votre container est immédiatement fonctionnel.
Remarquez que si l'image `ubuntu:18.04` n'est pas disponible en local, elle est téléchargée.

Pour afficher la liste des containers de notre l'hôte

```
# lxc list
+--------+---------+----------------------+------+-----------+-------------+
|  NAME  |  STATE  |         IPV4         | IPV6 |   TYPE    | SNAPSHOTS |
+--------+---------+----------------------+------+-----------+-------------+
| ubuntu | RUNNING | 10.121.162.194 (eth0) |      | CONTAINER | 0         |
+--------+---------+----------------------+------+-----------+-------------+
```

Notez qu'il est possible de configurer les colonnes à afficher :

```
#  lxc list -c
nsSm4,volatile.eth0.hwaddr:MAC,config:image.os,devices:eth0.parent
:ETHP
```



Pour se connecter avec un bash, tapez la commande:

```
sudo lxc exec ubuntu bash
root@ubuntu:~#
```

Vous avez maintenant accès à un Shell depuis l'intérieur de votre container.
Affichez l'adresse ip de votre container:

```
root@ubuntu:~# ip a
root@ubuntu:~# exit
```

Faisons un petit récapitulatif. Nous avons créé un container ubuntu à partir de l'image `ubuntu:18.04` avec la commande `lxc launch`. Puisque l'image `ubuntu:18.04`

n'était pas disponible en local, elle a été téléchargée. On a pu vérifier que le conteneur une fois installé, était directement utilisable. La commande `exit` nous permet de sortir du conteneur sans pour autant l'éteindre. Les commandes , `lxc list` ou `lxc-ls` permettent de vérifier l'état de notre container.

Il nous faut vérifier maintenant les différents fichiers qui constituent notre container. Pour cela rendez-vous dans le répertoire /var/lib/lxc/. Affichez son contenu avec la commande ls. C'est ici que se trouvent les fichiers qui constituent votre container. Ouvrez le répertoire Ubuntu et affichez son contenu. On y trouve un fichier `config`, qui contient les différents éléments de configuration de votre conteneur, et le répertoire rootfs qui correspond au système de fichiers de votre conteneur.

Mise en place d'un serveur web dans le container ubuntu

Relancez le container `ubuntu` . Une fois à l'intérieur, mettez à jour apt.

```
root@ubuntu:~# apt update
```

Upgradez votre container

```
root@ubuntu:~# apt upgrade
```

Installez apache2 dans notre container

```
root@ubuntu:~# apt install apache2
```

Vérifiez que apache soit lancé correctement

```
root@ubuntu:~# netstat -plnt
Proto  Recv-Q Send-Q Local Address  Foreign Address              State          PID/Program
name
tcp    0      0      127.0.0.53:53  0.0.0.0:*             LISTEN
172/systemd-resolve
tcp    0      0      0.0.0.0:22     0.0.0.0:*             LISTEN         227/sshd
tcp6   0      0      :::80          :::*                  LISTEN         1748/apache2
tcp6   0      0      :::22          :::*                  LISTEN         227/sshd
```

Test votre serveur apache

```
root@ubuntu:~# curl 127.0.0.1
```

Le code HTML de votre page index.html devrait s'afficher.

Gestion des fichiers

Copie de fichier du container vers l'hôte

Créez un fichier vide "fichier1" à la racine de votre utilisateur root du container ubuntu.

```
root@ubuntu:~# touch fichier1
root@ubuntu:~# ls
```

Sortez du container ubuntu

```
root@ubuntu:~# exit
```

Pour sortir le fichier "`fichier1`" du container ubuntu vers l'hôte.
`lxc file pull /ubuntu/root/fichier1 /home/&USER`

Copie de fichier de l'hôte vers un container

Sur l'hôte, créez un fichier index1.html
`echo "<H1>Coucou</H1>" > ./index1.html`

Pour copier le fichier index1.html présent sur l'hôte vers le container ubuntu: **`sudo lxc file push ./index.html ubuntu/var/www/html/index1.html`**
Pour vérifier que notre fichier a bien été copié à la racine de notre serveur Web.

```
sudo lxc exec ubuntu bash
root@ubuntu:~#
root@ubuntu:~# cd /var/www/html/
root@ubuntu:~# ls
root@ubuntu:~# exit
```

Rendre le serveur web accessible

Les périphériques proxy autorisent le transfert des connexions réseau entre l'hôte et une instance (un container). Ceci permet au trafic réseau atteignant l'adresse IP de notre hôte d'être redirigé vers l'adresse IP interne d'un container.
Ici, nous allons rédiger les connexions sur le port 80 de notre hôte vers le port 80 du container ubuntu et rendre accessible notre serveur apache.

```
sudo lxc config device add ubuntu myport80 proxy listen=tcp:0.0.0.0:80
connect=tcp:127.0.0.1:80
```

Cette commande a créé un périphérique proxy attaché au container *ubuntu.*

1. **lxc config device add**, ajoute un périphérique ,
2. **ubuntu**, au container ubuntu,
3. **myport80**, portant le nom myport80,
4. **proxy** de type proxy.
5. **listen=tcp:0.0.0.0:80**, a l'écoute des requêtes TCP sur le port 80 (sur l'hôte) en provenance de tous les réseaux
6. **connect=tcp:127.0.0.1:80**, que nous connectons au le port 80 de l'adresse loopback du container, où devrait répondre notre serveur Apache.

Référez vous à la documentation "device proxy " pour plus d'information :
https://linuxcontainers.org/lxd/docs/master/instances#type-proxy

Toujours depuis l'hôte ouvrez l'URL http://127.0.0.1:80. La page par défaut du serveur web s'affiche.

Maintenant, ouvronscette url: http://127.0.0.1:80/index1.html
Voici le fichier que nous avons transféré précédemment.

Configuration réseau de l'hôte LXD

La virtualisation réseau de LXC est basée sur le composant lxc-net et repose sur l'usage de ponts, c'est-à-dire de la couche 2. Un pont agit comme un commutateur virtuel reliant une interface réseau physique (hôte) avec une interface virtuelle (container) .

Configuration réseau des conteneurs

créer une interface eth1 réseau sur ubuntu liée au bridge de l'hôte. Sur l'hôte :

```
# lxc config device add ubuntu eth1 nic name=eth1 nictype=bridged
parent=virbr0
```

Cette commande ajoute une carte réseau (nic) nommée eth0 (name) qui va s'attacher sur un bridge (nictype=bridged) nommé virbr0 (parent=virbr0). Retournons sur apollo :

```
# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN group default qlen 1000
...
5: eth1@if6: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN
group default qlen 1000
...
```

Tentons de récupérer un bail DHCP et de pinger Google :

```
# dhclient
# ping www.google.fr
PING www.google.fr (172.217.19.227) 56(84) bytes of data.
64 bytes from par21s11-in-f3.1e100.net (172.217.19.227):
icmp_seq=1 ttl=52 time=2.48 ms
64 bytes from par21s11-in-f3.1e100.net (172.217.19.227):
icmp_seq=2 ttl=52 time=2.16 ms
```

Container privilégié et conteneur non privilégié

Sauvegardes et restauration d'un conteneur

Sur l'hôte, vos conteneurs sont stockés dans le répertoire /var/lib/lxc/<Nom du conteneur>.

Rendez-vous dans ce répertoire et affichez la liste des containers:

```
cd /var/lib/lxc/
ls
```

Vous devez y trouver le répertoire « ubuntu », répertoire où est stocké votre container ubuntu.
Ouvrez le répertoire `ubuntu`. On n'y trouve au minimum le fichier « config » et répertoire « rootfs ». La sauvegarde d'un container se limite à l'archivage du fichier config et du répertoire rootfs. Il est tout à fait possible de réaliser cette opération avec la commande tar, cependant la commande `lxc-snapshot` s'en chargera bien mieux que nous. Notez que cette commande ne peut être exécutée que si votre container est arrêté.

```
lxc-stop ubuntu
lxc-snapshot ubuntu
```

Afficher de nouveau le contenu du répertoire /var/lib/lxc/ubuntu

Un répertoire `snaps` a été créé par la commande `lxc-snapshot`. Dans ce nouveau dossier un répertoire `snap0` est présent. Il contient votre premier instantané du container `ubuntu`.

Afficher le contenu du répertoire /var/lib/lxc/ubuntu/snaps/snap0
Il contient le fichier `config`, le répertoire `rootfs` et le fichier `ts`. Le fichier `ts` contient uniquement la date et l'heure à laquelle le snapshot a été réalisé.

Réaliser à nouveau un snapshot du conteneur Ubuntu avec la commande `lxc-snapshot ubuntu`.

Un répertoire snap1 a été créé dans `/var/lib/lxc/ubuntu/snaps/`.

Pour restaurer un container à partir d'un snapshot il suffit de mentionner l'option –r <non du snapshot> , l'option -n <nom du conteneur> et l'option –N <nom du nouveau container>.

```
lxc-snapshot -r snap0 -n ubuntu -N ubuntu_snap0
```

Pour plus détails, référez vous au man de lxc-snapshot

Récapitulatif des commandes

lxc-attach
Exécute une commande dans un conteneur. Si aucune commande n'est mentionnée, le shell par défaut de l'utilisateur sera exécuté. Cette commande ne peut être utilisée que sur un container démarré.

lxc-autostart
Permet de rebooter, tuer ou arrêter les containers possedant l'option `lxc.start.auto` dans leur fichier « `config` ».

lxc-console
Ouvre une session dans un conteneur spécifique. Cette commande implique que votre container soit capable de gérer les logins. En cas de perte de connexion, relancer cette commande et la session reprendra là où elle en était.

lxc-execute
Démarre le conteneur spécifié et exécute la commande passée en paramètres. Lorsque la commande sera terminée, le conteneur sera stoppé.

lxc-create
Créer un nouveau container.

lxc-destroy
Détruit définitivement un container.

lxc-start
Démarre le conteneur spécifié

lxc-stop
Arrête le conteneur spécifié

lxc-ls
Afficher les conteneurs existant. L'option au -f affichera également leurs états.

lxc-info
Affiche les informations détaillées du conteneur spécifié.

lxc-freeze
Gèle tous les processus du conteneur spécifié. Ses processus resteront stoppés tant que la commande lxc-unfreeze ne sera pas exécutée sur ce conteneur.

lxc-unfreeze
Débloque les processus du conteneur spécifié où la commande lxc-freeze avait été exécutée.

lxc-snapshot
Créer un instantané de sauvegarde d'un conteneur.

lxc-copy
Permet de créer une copie complète d'un container ou un instantané.

lxc-cgroup

Permet de manipuler à chaud les groupes de contrôle d'un container.

lxc-device

Permet de rajouter à chaud des interfaces un container.

lxc-usernexec

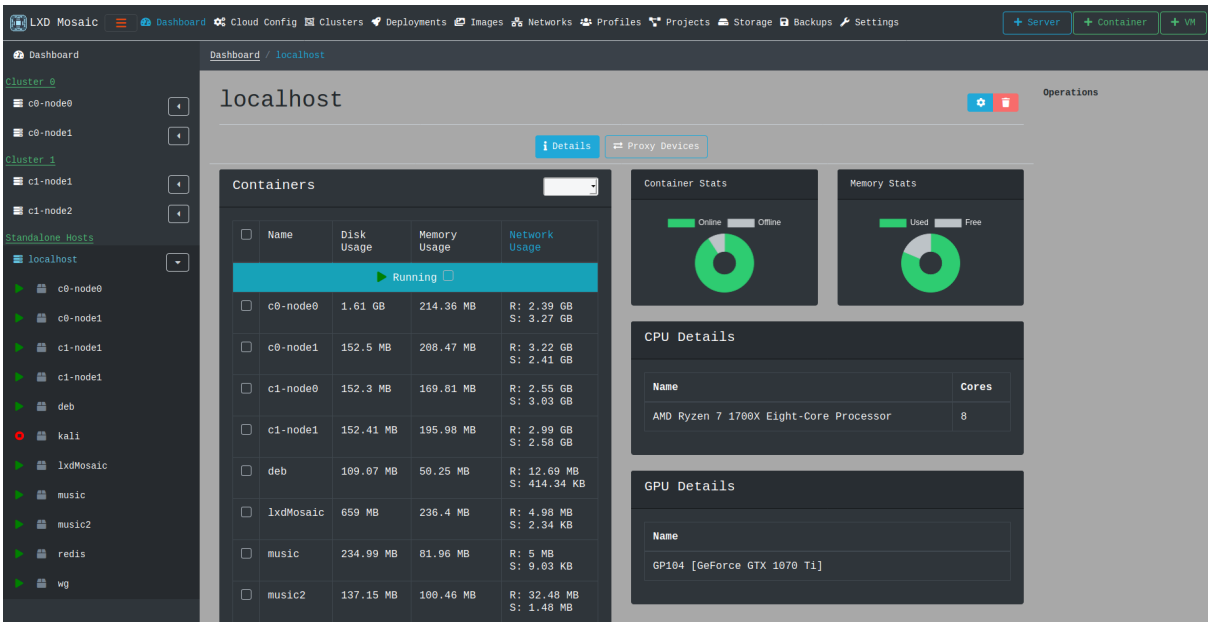
Permet d'exécuter des commandes avec les privilèges root sur un conteneur non privilégié.

lxc-wait

Cette commande permet d'exécuter un script lorsque le conteneur spécifié aura atteint un certain état (stop, running, ...).

TP3: Gestion d'un serveur LXD

Installez l'interface web de gestion LXdMosaic:




```
# snap install lxdmosaic
```

En cas dé soucis, choisissez une méthode adaptée à votre host et vos privilèges.
https://github.com/turtle0x1/LxdMosaic

La documentation de LxdMosaic est disponible ici:
https://lxdmosaic.readthedocs.io/en/latest/

Création d'un cluster LXD

Documentation : https://linuxcontainers.org/lxd/docs/master/clustering

Copier un container sur un autre node

Orchestration