Version : **2022.01**

Dernière mise-à-jour : 2021/12/29 10:32

DOF101 - La Virtualisation par Isolation

Contenu du Module

- **DOF101 - La Virtualisation par Isolation**
  - Contenu du Module
  - Présentation de la Virtualisation par Isolation
    - Historique
  - Présentation des Namespaces
  - Présentation des CGroups
    - LAB #1 - Travailler avec les CGroups
      - 1.1 - Limitation de la Mémoire
      - 1.2 - Le Paquet cgroup-tools
        - La commande cgcreate
        - La Commande cgexec
        - La Commande cgdelete
        - Le Fichier /etc/cgconfig.conf
  - Présentation de Linux Containers
    - LAB #2 - Travailler avec LXC
      - 2.1 - Installation
      - 2.2 - Création d'un Conteneur Simple
      - 2.3 - Démarrage d'un Conteneur Simple
      - 2.4 - S'attacher à un Conteneur Simple
      - 2.5 - Commandes LXC de Base
        - La Commande lxc-console
        - La Commande lxc-stop
        - La Commande lxc-execute

    - La Commande lxc-info
    - La Commande lxc-freeze
    - La Commande lxc-unfreeze
    - Autres commandes
- 2.6 - Création d'un Conteneur Non-Privilégié
    - User Namespaces
    - Création d'un Utilisateur Dédié
    - Création du Mappage
    - Création du Conteneur
    - Contrôle du Mappage
- 2.7 - Création d'un Conteneur Éphémère
    - La Commande lxc-copy
- 2.8 - Sauvegarde des Conteneurs
    - La Commande lxc-snapshot

Présentation de la Virtualisation par Isolation

Un isolateur est un logiciel qui permet d'isoler l'exécution des applications dans des **containers**, des **contextes** ou des **zones d'exécution**.

Historique

- **1979** - chroot - l'isolation par changement de racine,
- **2000** - BSD Jails - l'isolation en espace utilisateur,
- **2004** - Solaris Containers - l'isolation par **zones**,
- **2005** - OpenVZ - l'isolation par **partitionnement du noyau** sous Linux,
- **2008** - LXC - LinuX Containers - l'isolation en utilisant des **namespaces** et des **CGroups** avec **liblxc**,
- **2013** - Docker - l'isolation en utilisant des **namespaces** et des **CGroups** avec **libcontainer**,
- **2014** - LXD - LinuX Container Daemon - l'isolation en utilisant des **namespaces** et des **CGroups** avec **liblxc**.

Présentation des Namespaces

Les espaces de noms permettent de regrouper des processus dans un même espace et d'attribuer des droits sur des ressources par espace. Ceci permet l'exécution de plusieurs init, chacun dans un namespace, afin de recréer un environnement pour les processus qui doivent être isolés.

Présentation des CGroups

Les Groupes de Contrôles (Control Groups) aussi appelés **CGroups**, sont une nouvelle façon de contrôler et de limiter des ressources. Les groupes de contrôle permettent l'allocation de ressources, même d'une manière dynamique pendant que le système fonctionne, telles le temps processeur, la mémoire système, la bande réseau, ou une combinaison de ces ressources parmi des groupes de tâches (processus) définis par l'utilisateur et exécutés sur un système.

Les CGroups sont organisés de manière hiérarchique, comme des processus. Par contre, la comparaison entre les deux démontre que tandis que les processus se trouvent dans une arborescence unique descendant tous du processus init et héritant de l'environnement de leurs parents, les CGroups peuvent être multiples donnant lieu à des arborescences ou **hiérarchies** multiples qui héritent de certains attributs de leurs groupes de contrôle parents.

Ces hiérarchies multiples et séparés sont necéssaires parce que chaque hiérarchie est attaché à un ou plusieurs sous-système(s) aussi appelés des **Contrôleurs de Ressources** ou simplement des **Contrôleurs**. Les contrôleurs disponibles sont :

- **blkio** - utilisé pour établir des limites sur l'accès des entrées/sorties à partir et depuis des périphériques blocs,
- **cpu** - utilisé pour fournir aux tâches des groupes de contrôle accès au CPU grâce au planificateur,
- **cpuacct** - utilisé pour produire des rapports automatiques sur les ressources CPU utilisées par les tâches dans un groupe de contrôle,
- **cpuset** - utilisé pour assigner des CPU individuels sur un système multicœur et des nœuds de mémoire à des tâches dans un groupe de contrôle,
- **devices** - utilisé pour autoriser ou pour refuser l'accès des tâches aux périphériques dans un groupe de contrôle,
- **freezer** - utilisé pour suspendre ou pour réactiver les tâches dans un groupe de contrôle,
- **memory** - utilisé pour établir les limites d'utilisation de la mémoire par les tâches d'un groupe de contrôle et pour génèrer des rapports automatiques sur les ressources mémoire utilisées par ces tâches,
- **net_cls** - utilisé pour repérer les paquets réseau avec un identifiant de classe (classid) afin de permettre au contrôleur de trafic Linux, **tc**, d'identifier les paquets provenant d'une tâche particulière d'un groupe de contrôle.
- **perf_event** - utilisé pour permettre le monitoring des CGroups avec l'outil perf,

- **hugetlb** - utilisé pour limiter des ressources sur des pages de mémoire virtuelle de grande taille.

Les hiérarchies ont des points de montage dans le répertoire **/sys/fs/cgroup** :

```
trainee@debian9:~$ su -
Mot de passe : fenestros
root@debian9:~# ls -l /sys/fs/cgroup/
total 0
dr-xr-xr-x 2 root root  0 mai   26 08:52 blkio
lrwxrwxrwx 1 root root 11 mai   26 08:52 cpu -> cpu,cpuacct
lrwxrwxrwx 1 root root 11 mai   26 08:52 cpuacct -> cpu,cpuacct
dr-xr-xr-x 2 root root  0 mai   26 08:52 cpu,cpuacct
dr-xr-xr-x 2 root root  0 mai   26 08:52 cpuset
dr-xr-xr-x 5 root root  0 mai   26 08:52 devices
dr-xr-xr-x 2 root root  0 mai   26 08:52 freezer
dr-xr-xr-x 2 root root  0 mai   26 08:52 memory
lrwxrwxrwx 1 root root 16 mai   26 08:52 net_cls -> net_cls,net_prio
dr-xr-xr-x 2 root root  0 mai   26 08:52 net_cls,net_prio
lrwxrwxrwx 1 root root 16 mai   26 08:52 net_prio -> net_cls,net_prio
dr-xr-xr-x 2 root root  0 mai   26 08:52 perf_event
dr-xr-xr-x 5 root root  0 mai   26 08:52 pids
dr-xr-xr-x 5 root root  0 mai   26 08:52 systemd
```

Systemd organise les processus dans chaque CGroup. Par exemple tous les processus démarrés par le serveur Apache se trouveront dans le même CGroup, y compris les scripts CGI. Ceci implique que la gestion des ressources en utilisant des hiérarchies est couplé avec l'arborescence des unités de Systemd.

En haut de l'arborescence des unités de Systemd se trouve la tranche root - **-.slice**, dont dépend :

- le **system.slice** - l'emplacement des services système,
- le **user.slice** - l'emplacement des sessions des utilisateurs,
- le **machine.slice** - l'emplacement des machines virtuelles et conteneurs.

En dessous des tranches peuvent se trouver :

- des **scopes** - des processus crées par **fork**,
- des **services** - des processus créés par une **Unité**.

Les slices peuvent être visualisés avec la commande suivante :

```
root@debian9:~# systemctl list-units --type=slice
UNIT               LOAD   ACTIVE SUB    DESCRIPTION
-.slice            loaded active active Root Slice
system-getty.slice loaded active active system-getty.slice
system.slice       loaded active active System Slice
user-1000.slice    loaded active active User Slice of trainee
user-112.slice     loaded active active User Slice of lightdm
user.slice         loaded active active User and Session Slice

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

6 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
```

L'arborescence des unités de Systemd est la suivante :

```
root@debian9:~# systemd-cgls
Control group /:
-.slice
├─user.slice
│ ├─user-112.slice
│ │ ├─user@112.service
│ │ │ ├─dbus.service
│ │ │ │ └─539 /usr/bin/dbus-daemon --session --address=systemd: --nofork --nopidfile --systemd-activation
│ │ │ ├─init.scope
│ │ │ │ ├─527 /lib/systemd/systemd --user
│ │ │ │ └─528 (sd-pam)
```

```
│ │ │ └─gvfs-daemon.service
│ │ │   └─547 /usr/lib/gvfs/gvfsd
│ │ └─session-c1.scope
│ │   ├─524 lightdm --session-child 18 21
│ │   ├─532 /usr/sbin/lightdm-gtk-greeter
│ │   ├─534 /usr/lib/at-spi2-core/at-spi-bus-launcher --launch-immediately
│ │   ├─541 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print-
address 3
│ │   └─543 /usr/lib/at-spi2-core/at-spi2-registryd --use-gnome-session
│ └─user-1000.slice
│   ├─session-2.scope
│   │ ├─668 sshd: trainee [priv]
│   │ ├─679 sshd: trainee@pts/0
│   │ ├─680 -bash
│   │ ├─689 su -
│   │ ├─690 -su
│   │ ├─708 systemd-cgls
│   │ └─709 systemd-cgls
│   └─user@1000.service
│     └─init.scope
│       ├─670 /lib/systemd/systemd --user
│       └─671 (sd-pam)
├─init.scope
│ └─1 /sbin/init
└─system.slice
  ├─lightdm.service
  │ ├─410 /usr/sbin/lightdm
  │ ├─425 /usr/lib/xorg/Xorg :0 -seat seat0 -auth /var/run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch
  │ └─588 lightdm --session-child 14 21
  ├─anacron.service
lines 1-39
```

En utilisant Systemd, plusieurs ressources peuvent être limitées :

- **CPUShares** - par défaut 1024,
- **MemoryLimit** - limite exprimée en Mo ou en Go. Pas de valeur par défaut,
- **BlockIOWeight** - valeur entre 10 et 1000. Pas de valeur par défaut,
- **StartupCPUShares** - comme CPUShares mais uniquement appliqué pendant le démarrage,
- **StartupBlockIOWeight** - comme BlockIOWeight mais uniquement appliqué pendant le démarrage,
- **CPUQuota** - utilisé pour limiter le temps CPU, même quand le système ne fait rien.

Important : Consultez le manuel systemd.resource-control(5) pour voir les paramètres CGroup qui peuvent être passés à systemctl.

LAB #1 - Travailler avec les CGroups

1.1 - Limitation de la Mémoire

Pour travailler avec les CGroups dans Debian 9, il convient d'installer les outils nécessaires :

```
root@debian9:~# apt-get install libcgroup1 cgroup-tools
```

Commencez ensuite par créer le script **hello-world.sh** qui servira à générer un processus pour travailler avec les CGroups :

```
root@debian9:~# vi hello-world.sh
root@debian9:~# cat hello-world.sh
#!/bin/bash
while [ 1 ]; do
    echo "hello world"
    sleep 60
done
```

Rendez le script exécutable et testez-le :

```
root@debian9:~# chmod u+x hello-world.sh
root@debian9:~# ./hello-world.sh
hello world
hello world
^C
```

Créez maintenant un CGroup dans le sous-système **memory** appelé **helloworld** :

```
root@debian9:~# mkdir /sys/fs/cgroup/memory/helloworld
```

Par défaut, ce CGroup héritera de l'ensemble de la mémoire disponible. Pour éviter cela, créez maintenant une limite de **40000000** octets pour ce CGroup :

```
root@debian9:~# echo 40000000 > /sys/fs/cgroup/memory/helloworld/memory.limit_in_bytes
root@debian9:~# cat /sys/fs/cgroup/memory/helloworld/memory.limit_in_bytes
39997440
```

Important - Notez que les 40 000 000 demandés sont devenus 39 997 440 ce qui correspond à un nombre entier de pages mémoire du noyau de 4Ko. (39 997 440 / 4096 = 9 765).

Lancez maintenant le script **helloworld.sh** :

```
root@debian9:~# ./hello-world.sh &
[1] 1012
root@debian9:~# hello world

root@debian9:~# ps aux | grep hello-world
root      1012  0.0  0.1  11172  2868 pts/0    S    10:24   0:00 /bin/bash ./hello-world.sh
root      1015  0.0  0.0  12784   968 pts/0    S+   10:25   0:00 grep hello-world
```

Notez l'héritage par défaut :

```
root@debian9:~# ps -ww -o cgroup 1012
CGROUP
6:devices:/user.slice,3:pids:/user.slice/user-1000.slice/session-2.scope,1:name=systemd:/user.slice/user-1000.slice/session-2.scope
```

Insérer le PID de notre script dans le CGroup **helloworld** :

```
root@debian9:~# echo 1012 > /sys/fs/cgroup/memory/helloworld/cgroup.procs
```

Notez maintenant l'héritage de la limitation de la mémoire - **2:memory:/helloworld** :

```
root@debian9:~# ps -ww -o cgroup 1012
CGROUP
6:devices:/user.slice,3:pids:/user.slice/user-1000.slice/session-2.scope,2:memory:/helloworld,1:name=systemd:/user.slice/user-1000.slice/session-2.scope
```

Constatez ensuite l'occupation mémoire réelle :

```
root@debian9:~# cat /sys/fs/cgroup/memory/helloworld/memory.usage_in_bytes
319488
```

Tuez le script **hello-world.sh** :

```
root@debian9:~# kill 1012
root@debian9:~# ps aux | grep hello-world
root      1038  0.0  0.0  12784   936 pts/0    S+   10:33   0:00 grep hello-world
[1]+  Complété              ./hello-world.sh
```

Créez un second CGroup beaucoup plus restrictif :

```
root@debian9:~# mkdir /sys/fs/cgroup/memory/helloworld1
root@debian9:~# echo 6000 > /sys/fs/cgroup/memory/helloworld1/memory.limit_in_bytes
root@debian9:~# cat /sys/fs/cgroup/memory/helloworld1/memory.limit_in_bytes
```

```
4096
```

Relancez le script **hello-world.sh** et insérez-le dans le nouveau CGroup :

```
root@debian9:~# ./hello-world.sh &
[1] 1042
root@debian9:~# hello world

root@debian9:~# echo 1042 > /sys/fs/cgroup/memory/helloworld1/cgroup.procs
```

Attendez la prochaine sortie de **hello world** sur le canal standard puis constatez que le script s'arrête :

```
root@debian9:~# hello world

[1]+  Processus arrêté      ./hello-world.sh
```

Consultez en suite la fin du fichier **/var/log/messages** :

```
root@debian9:~# tail /var/log/messages
Jul 28 10:39:26 debian9 kernel: [ 1501.775169]  [<ffffffffabe6253d>] ? __do_page_fault+0x4bd/0x4f0
Jul 28 10:39:26 debian9 kernel: [ 1501.775171]  [<ffffffffabfc2950>] ? SyS_brk+0x160/0x180
Jul 28 10:39:26 debian9 kernel: [ 1501.775175]  [<ffffffffac41a358>] ? page_fault+0x28/0x30
Jul 28 10:39:26 debian9 kernel: [ 1501.775176] Task in /helloworld1 killed as a result of limit of /helloworld1
Jul 28 10:39:26 debian9 kernel: [ 1501.775180] memory: usage 4kB, limit 4kB, failcnt 17
Jul 28 10:39:26 debian9 kernel: [ 1501.775180] memory+swap: usage 0kB, limit 9007199254740988kB, failcnt 0
Jul 28 10:39:26 debian9 kernel: [ 1501.775181] kmem: usage 0kB, limit 9007199254740988kB, failcnt 0
Jul 28 10:39:26 debian9 kernel: [ 1501.775181] Memory cgroup stats for /helloworld1: cache:0KB rss:4KB
rss_huge:0KB mapped_file:0KB dirty:0KB writeback:0KB inactive_anon:0KB active_anon:0KB inactive_file:0KB
active_file:0KB unevictable:0KB
Jul 28 10:39:26 debian9 kernel: [ 1501.775188] [ pid ]   uid  tgid total_vm      rss nr_ptes nr_pmds swapents
oom_score_adj name
Jul 28 10:39:26 debian9 kernel: [ 1501.775219] [ 1042]     0  1042     2795      732      10       3        0
0 hello-world.sh
```

Important - Notez la trace **Task in /helloworld1 killed as a result of limit of /helloworld1**.

1.2 - Le Paquet cgroup-tools

Le paquet **cgroup-tools** installe des commandes dites *de facilité* dont :

La commande cgcreate

Cette commande permet la création d'un CGroup :

```
root@debian9:~# cgcreate -g memory:helloworld2
root@debian9:~# ls -l /sys/fs/cgroup/memory/helloworld2/
total 0
-rw-r--r-- 1 root root 0 juil. 28 11:09 cgroup.clone_children
--w--w--w- 1 root root 0 juil. 28 11:09 cgroup.event_control
-rw-r--r-- 1 root root 0 juil. 28 11:09 cgroup.procs
-rw-r--r-- 1 root root 0 juil. 28 11:09 memory.failcnt
--w------- 1 root root 0 juil. 28 11:09 memory.force_empty
-rw-r--r-- 1 root root 0 juil. 28 11:09 memory.kmem.failcnt
-rw-r--r-- 1 root root 0 juil. 28 11:09 memory.kmem.limit_in_bytes
-rw-r--r-- 1 root root 0 juil. 28 11:09 memory.kmem.max_usage_in_bytes
-r--r--r-- 1 root root 0 juil. 28 11:09 memory.kmem.slabinfo
-rw-r--r-- 1 root root 0 juil. 28 11:09 memory.kmem.tcp.failcnt
-rw-r--r-- 1 root root 0 juil. 28 11:09 memory.kmem.tcp.limit_in_bytes
-rw-r--r-- 1 root root 0 juil. 28 11:09 memory.kmem.tcp.max_usage_in_bytes
-r--r--r-- 1 root root 0 juil. 28 11:09 memory.kmem.tcp.usage_in_bytes
-r--r--r-- 1 root root 0 juil. 28 11:09 memory.kmem.usage_in_bytes
-rw-r--r-- 1 root root 0 juil. 28 11:09 memory.limit_in_bytes
```

```
-rw-r--r-- 1 root root 0 juil. 28 11:09 memory.max_usage_in_bytes
-rw-r--r-- 1 root root 0 juil. 28 11:09 memory.move_charge_at_immigrate
-r--r--r-- 1 root root 0 juil. 28 11:09 memory.numa_stat
-rw-r--r-- 1 root root 0 juil. 28 11:09 memory.oom_control
---------- 1 root root 0 juil. 28 11:09 memory.pressure_level
-rw-r--r-- 1 root root 0 juil. 28 11:09 memory.soft_limit_in_bytes
-r--r--r-- 1 root root 0 juil. 28 11:09 memory.stat
-rw-r--r-- 1 root root 0 juil. 28 11:09 memory.swappiness
-r--r--r-- 1 root root 0 juil. 28 11:09 memory.usage_in_bytes
-rw-r--r-- 1 root root 0 juil. 28 11:09 memory.use_hierarchy
-rw-r--r-- 1 root root 0 juil. 28 11:09 notify_on_release
-rw-r--r-- 1 root root 0 juil. 28 11:09 tasks
```

Il n'existe cependant pas de commande pour affecter une limitation de la mémoire :

```
root@debian9:~# echo 40000000 > /sys/fs/cgroup/memory/helloworld2/memory.limit_in_bytes
```

La Commande cgexec

Cette commande permet d'insérer la limitation dans le CGroup **et** de lancer le script en une seule ligne :

```
root@debian9:~# cgexec -g memory:helloworld2 ./hello-world.sh &
[2] 1860
root@debian9:~# hello world

root@debian9:~#
```

La Commande cgdelete

Une fois le script terminé, cette commande permet de supprimer le cgroup :

```
root@debian9:~# ps aux | grep *.sh
root      1073  0.0  0.1  11172  2868 pts/0    S    11:18   0:00 /bin/bash ./hello-world.sh
root      1076  0.0  0.0    528     4 pts/0    R+   11:18   0:00 grep hello-world.sh
root@debian9:~# kill 1073
root@debian9:~# ps aux | grep *.sh
root      1078  0.0  0.0  12784   920 pts/0    S+   11:18   0:00 grep hello-world.sh
[1]+  Complété              cgexec -g memory:helloworld2 ./hello-world.sh
root@debian9:~# cgdelete memory:helloworld2
root@debian9:~# ls -l /sys/fs/cgroup/memory/helloworld2/
ls: impossible d'accéder à '/sys/fs/cgroup/memory/helloworld2/': Aucun fichier ou dossier de ce type
```

Le Fichier /etc/cgconfig.conf

Afin de les rendre persistants, il convient de créer les CGroups dans le fichier **/etc/cgconfig.conf** :

```
root@debian9:~# vi /etc/cgconfig.conf
root@debian9:~# cat /etc/cgconfig.conf
group helloworld2 {
    cpu {
        cpu.shares = 100;
    }
    memory {
        memory.limit_in_bytes = 40000;
    }
}
```

Important - Notez la création de **deux** limitations, une de 40 000 octets de mémoire et l'autre de **100 cpu.shares**. Cette dernière est une valeur exprimée sur 1 024, où 1 024 représente 100% du temps CPU. La limite fixée est donc equivalente à 9,77% du temps CPU.

Créez donc les deux CGroups concernés :

```
root@debian9:~# cgcreate -g memory:helloworld2
root@debian9:~# ls -l /sys/fs/cgroup/memory/helloworld2/
total 0
-rw-r--r-- 1 root root 0 juil. 28 12:47 cgroup.clone_children
--w--w--w- 1 root root 0 juil. 28 12:47 cgroup.event_control
-rw-r--r-- 1 root root 0 juil. 28 12:47 cgroup.procs
-rw-r--r-- 1 root root 0 juil. 28 12:47 memory.failcnt
--w------- 1 root root 0 juil. 28 12:47 memory.force_empty
-rw-r--r-- 1 root root 0 juil. 28 12:47 memory.kmem.failcnt
-rw-r--r-- 1 root root 0 juil. 28 12:47 memory.kmem.limit_in_bytes
-rw-r--r-- 1 root root 0 juil. 28 12:47 memory.kmem.max_usage_in_bytes
-r--r--r-- 1 root root 0 juil. 28 12:47 memory.kmem.slabinfo
-rw-r--r-- 1 root root 0 juil. 28 12:47 memory.kmem.tcp.failcnt
-rw-r--r-- 1 root root 0 juil. 28 12:47 memory.kmem.tcp.limit_in_bytes
-rw-r--r-- 1 root root 0 juil. 28 12:47 memory.kmem.tcp.max_usage_in_bytes
-r--r--r-- 1 root root 0 juil. 28 12:47 memory.kmem.tcp.usage_in_bytes
-r--r--r-- 1 root root 0 juil. 28 12:47 memory.kmem.usage_in_bytes
-rw-r--r-- 1 root root 0 juil. 28 12:47 memory.limit_in_bytes
-rw-r--r-- 1 root root 0 juil. 28 12:47 memory.max_usage_in_bytes
-rw-r--r-- 1 root root 0 juil. 28 12:47 memory.move_charge_at_immigrate
-r--r--r-- 1 root root 0 juil. 28 12:47 memory.numa_stat
-rw-r--r-- 1 root root 0 juil. 28 12:47 memory.oom_control
---------- 1 root root 0 juil. 28 12:47 memory.pressure_level
-rw-r--r-- 1 root root 0 juil. 28 12:47 memory.soft_limit_in_bytes
-r--r--r-- 1 root root 0 juil. 28 12:47 memory.stat
-rw-r--r-- 1 root root 0 juil. 28 12:47 memory.swappiness
-r--r--r-- 1 root root 0 juil. 28 12:47 memory.usage_in_bytes
-rw-r--r-- 1 root root 0 juil. 28 12:47 memory.use_hierarchy
-rw-r--r-- 1 root root 0 juil. 28 12:47 notify_on_release
-rw-r--r-- 1 root root 0 juil. 28 12:47 tasks
```

```
root@debian9:~# cgcreate -g cpu:helloworld2
root@debian9:~# ls -l /sys/fs/cgroup/cpu/helloworld2/
total 0
```

```
-rw-r--r-- 1 root root 0 juil. 28 12:48 cgroup.clone_children
-rw-r--r-- 1 root root 0 juil. 28 12:48 cgroup.procs
-r--r--r-- 1 root root 0 juil. 28 12:48 cpuacct.stat
-rw-r--r-- 1 root root 0 juil. 28 12:48 cpuacct.usage
-r--r--r-- 1 root root 0 juil. 28 12:48 cpuacct.usage_all
-r--r--r-- 1 root root 0 juil. 28 12:48 cpuacct.usage_percpu
-r--r--r-- 1 root root 0 juil. 28 12:48 cpuacct.usage_percpu_sys
-r--r--r-- 1 root root 0 juil. 28 12:48 cpuacct.usage_percpu_user
-r--r--r-- 1 root root 0 juil. 28 12:48 cpuacct.usage_sys
-r--r--r-- 1 root root 0 juil. 28 12:48 cpuacct.usage_user
-rw-r--r-- 1 root root 0 juil. 28 12:48 cpu.cfs_period_us
-rw-r--r-- 1 root root 0 juil. 28 12:48 cpu.cfs_quota_us
-rw-r--r-- 1 root root 0 juil. 28 12:48 cpu.shares
-r--r--r-- 1 root root 0 juil. 28 12:48 cpu.stat
-rw-r--r-- 1 root root 0 juil. 28 12:48 notify_on_release
-rw-r--r-- 1 root root 0 juil. 28 12:48 tasks
```

Appliquez le contenu du fichier **/etc/cgconfig.conf** grâce à l'utilisation de la commande **cgconfigparser** :

```
root@debian9:~# cgconfigparser -l /etc/cgconfig.conf
root@debian9:~# cat /sys/fs/cgroup/memory/helloworld2/memory.limit_in_bytes
36864
root@debian9:~# cat /sys/fs/cgroup/cpu/helloworld2/cpu.shares
100
```

Présentation de Linux Containers

LAB #2 - Travailler avec LXC

2.1 - Installation

Les outils indispensables à l'utilisation des Linux Containers sous Debian sont inclus dans le paquet **lxc** :

```
root@debian9:~# apt update
root@debian9:~# apt install lxc
```

L'installation de ce paquet va créer les répertoires **/usr/share/lxc/config** contenant les fichiers de configurations des gabarits ainsi que le répertoire **/usr/share/lxc/templates** contenant fichiers de gabarits pour la création des conteneurs :

```
root@debian9:~# ls /usr/share/lxc
config  hooks  lxc.functions  lxc-patch.py  __pycache__  selinux  templates
root@debian9:~# ls /usr/share/lxc/config
alpine.common.conf     centos.userns.conf  debian.userns.conf      gentoo.userns.conf     oracle.common.conf
slackware.userns.conf      ubuntu-cloud.userns.conf
alpine.userns.conf     common.conf     fedora.common.conf       nesting.conf         oracle.userns.conf
sparclinux.common.conf      ubuntu.common.conf
archlinux.common.conf  common.conf.d       fedora.userns.conf       opensuse.common.conf  plamo.common.conf
sparclinux.userns.conf      ubuntu.lucid.conf
archlinux.userns.conf  common.seccomp      gentoo.common.conf       opensuse.userns.conf  plamo.userns.conf
ubuntu-cloud.common.conf  ubuntu.userns.conf
centos.common.conf     debian.common.conf  gentoo.moresecure.conf  openwrt.common.conf   slackware.common.conf
ubuntu-cloud.lucid.conf   userns.conf
root@debian9:~# ls /usr/share/lxc/templates
lxc-alpine    lxc-archlinux  lxc-centos  lxc-debian    lxc-fedora  lxc-openmandriva  lxc-oracle  lxc-slackware
lxc-sshd    lxc-ubuntu-cloud
lxc-altlinux  lxc-busybox    lxc-cirros  lxc-download  lxc-gentoo  lxc-opensuse      lxc-plamo   lxc-sparclinux
lxc-ubuntu
```

2.2 - Création d'un Conteneur Simple

Créez un conteneur simple en utilisant la commande suivante :

```
root@debian9:~# lxc-create -n lxc-bb -t busybox
```

```
chmod: impossible d'accéder à '/var/lib/lxc/lxc-bb/rootfs/bin/passwd': Aucun fichier ou dossier de ce type
setting root password to "root"
Failed to change root password
```

Important - Notez l'utilisation de l'option **-n** qui permet d'associer un nom au conteneur ainsi que l'option **-t** qui indique le gabarit à utiliser. Notez aussi que le gabarit est référencé par le nom du fichier dans le répertoire **/usr/share/lxc/templates** sans son préfix **lxc-**.

Le **backingstore** (*méthode de stockage*) utilisé par défaut est **dir** ce qui implique que le **rootfs** du conteneur se trouve sur disque dans le répertoire **/var/lib/lxc/** :

```
root@debian9:~# ls /var/lib/lxc/
lxc-bb

root@debian9:~# ls /var/lib/lxc/lxc-bb/
config  rootfs

root@debian9:~# ls /var/lib/lxc/lxc-bb/rootfs
bin  dev  etc  home  lib  lib64  mnt  proc  root  sbin  selinux  sys  tmp  usr    var
```

Il est à noter que LXC peut également utiliser des backingstores de type :

- ZFS
- Brtfs
- LVM
- Loop
- rbd (CephFS)

2.3 - Démarrage d'un Conteneur Simple

Pour démarrer le conteneur, il convient d'utiliser la commande **lxc-start** :

```
root@debian9:~# lxc-start --name lxc-bb
```

2.4 - S'attacher à un Conteneur Simple

Pour s'attacher au conteneur démarré, il convient d'utiliser la commande **lxc-attach** :

```
root@debian9:~# lxc-attach --name lxc-bb


BusyBox v1.22.1 (Debian 1:1.22.0-19+b3) built-in shell (ash)
Enter 'help' for a list of built-in commands.

~ # passwd
/bin/sh: passwd: not found
~ # which passwd
~ #
```

Important - Notez l'absence de la commande **passwd** dans le conteneur, ce qui explique l'erreur lors de la création de celui-ci.

Pour sortir du conteneur, il convient d'utiliser la commande **exit** ou bien la combinaison de touches **<Ctrl+d>** :

```
~ # [Ctrl+d]
~ # root@debian9:~#
```

Le fait de sortir du conteneur ne l'arrête pas pour autant, comme il peut être constaté par l'utilisation de la commande **lxc-ls** :

```
~ # root@debian9:~# [Entrée]
root@debian9:~# lxc-ls --running
lxc-bb
```

```
root@debian9:~# lxc-ls -f --running
NAME   STATE   AUTOSTART GROUPS IPV4 IPV6
lxc-bb RUNNING 0         -      -    -
```

2.5 - Commandes LXC de Base

La Commande lxc-console

Pour lancer une console attachée à un TTY dans le conteneur, il convient d'utiliser la commande **lxc-console** :

```
root@debian9:~# lxc-console --name lxc-bb

Connected to tty 1
                  Type <Ctrl+a q> to exit the console, <Ctrl+a Ctrl+a> to enter Ctrl+a itself

lxc-bb login: root
Password:
Login incorrect
lxc-bb login: trainee
Password:
Login incorrect
lxc-bb login:
```

Important - Notez que pour des raisons évidentes, le conteneur BusyBox ne sait pas gérer de logins.

Pour sortir de la console, il faut utiliser la combinaison de touches **<Ctrl+a> <q>** :

```
lxc-bb login: [Ctrl+a] [q] root@debian9:~#
```

La Commande lxc-stop

Pour arrêter le conteneur, utilisez la commande **lxc-stop** :

```
root@debian9:~# lxc-ls --running
lxc-bb
root@debian9:~# lxc-stop --name lxc-bb
root@debian9:~# lxc-ls --running
root@debian9:~#
```

La Commande lxc-execute

La commande **lxc-execute** démarre un conteneur (qui doit être créé mais arrêté), exécute la commande passée en argument grâce aux caractères **–** puis arrête le conteneur :

```
root@debian9:~# lxc-execute -n lxc-bb -- uname -a
init.lxc.static: initutils.c: mount_fs: 36 failed to mount /proc : Device or resource busy
Linux lxc-bb 4.9.0-8-amd64 #1 SMP Debian 4.9.130-2 (2018-10-27) x86_64 GNU/Linux
root@debian9:~# lxc-ls --running
root@debian9:~#
```

La Commande lxc-info

Cette commande donne des informations sur un conteneur :

```
root@debian9:~# lxc-info -n lxc-bb
Name:           lxc-bb
State:          STOPPED
```

La Commande lxc-freeze

La commande **lxc-freeze** met en pause tous les processus du conteneur :

```
root@debian9:~# lxc-start -n lxc-bb

root@debian9:~# lxc-ls --running
lxc-bb

root@debian9:~# lxc-info -n lxc-bb
Name:           lxc-bb
State:          RUNNING
PID:            3906
CPU use:        0.00 seconds
BlkIO use:      0 bytes
Memory use:     664.00 KiB
KMem use:       340.00 KiB

root@debian9:~# lxc-freeze -n lxc-bb

root@debian9:~# lxc-info -n lxc-bb
Name:           lxc-bb
State:          FROZEN
PID:            3906
CPU use:        0.00 seconds
BlkIO use:      0 bytes
Memory use:     664.00 KiB
KMem use:       340.00 KiB
```

La Commande lxc-unfreeze

La commande **lxc-unfreeze** annule l'effet d'une commande **lxc-freeze** précédente :

```
root@debian9:~# lxc-unfreeze -n lxc-bb

root@debian9:~# lxc-info -n lxc-bb
Name:           lxc-bb
State:          RUNNING
PID:            3906
CPU use:        0.00 seconds
BlkIO use:      0 bytes
Memory use:     664.00 KiB
KMem use:       340.00 KiB
```

Autres Commandes

Les autres commandes dont il faut avoir une connaissance sont :

Commande	Description
lxc-destroy	Permet de détruire complètement un conteneur
lxc-autostart	Permet de rebooter, tuer ou arrêter les conteneurs dont le drapeau **lxc.start.auto** est fixé dans le fichier **/var/lib/<nom_conteneur>/config**
lxc-cgroup	Permet de manipuler à chaud les CGroups pour un conteneur donné
lxc-device	Permet de rajouter à chaud les devices à un conteneur
lxc-usernsexec	Permet d'exécuter des commandes en tant que root dans un conteneur non-privilégié
lxc-wait	Permet d'attendre à ce qu'un conteneur ait atteint un certain état avant de continuer

2.6 - Création d'un Conteneur Non-Privilégié

User Namespaces

Un conteneur privilégié est un conteneur lancé par l'utilisateur **root** tandis qu'un conteneur non-privilégié est lancé par un utilisateur autre que root.
Un conteneur privilégié est moins sécurisé qu'un conteneur non-privilégié mais ne nécessite pas la gestion du mappage d'UID.

Le mappage d'UID est une fonctionnalité du noyau Linux appelée **user namespaces** ou encore **userns**. Cette fonctionnalité permet de mapper une plage d'UID de la machine hôte vers un utilisateur dont l'UID est 0 dans un autre espace de noms.

Création d'un Utilisateur Dédié

Commencez par créer l'utilisateur **lxcnp** dédié à la gestion des conteneurs non-privilégiés :

```
root@debian9:~# useradd -c "Utilisateur LXC Non-Priviligié" -s /bin/bash -m lxcnp
root@debian9:~# passwd lxcnp
Entrez le nouveau mot de passe UNIX : trainee
Retapez le nouveau mot de passe UNIX : trainee
passwd: password updated successfully
```

Les informations concernant le mappage des UID et GID se trouvent dans les fichiers **/etc/subuid** et **/etc/subgid** :

```
root@debian9:~# grep lxcnp /etc/sub{uid,gid}
/etc/subuid:lxcnp:165536:65536
/etc/subgid:lxcnp:165536:65536
```

Important - Ces informations indiquent que nous avons 65 536 UID et 65 536 GID disponibles pour le mappage vers l'UID et le GID **165 536**.

Création du Mappage

La valeur de **kernel.unprivileged_userns_clone** doit être **1**, or actuellement elle est de **0** :

```
root@debian9:~# cat /proc/sys/kernel/unprivileged_userns_clone
0
```

Fixez donc la valeur à **1** et appliquez la modification avec la commande **sysctl –system** :

```
root@debian9:~# echo "kernel.unprivileged_userns_clone=1" > /etc/sysctl.d/80-lxc-userns.conf
root@debian9:~# cat /etc/sysctl.d/80-lxc-userns.conf
kernel.unprivileged_userns_clone=1
root@debian9:~# sysctl --system
* Applying /etc/sysctl.d/80-lxc-userns.conf ...
kernel.unprivileged_userns_clone = 1
* Applying /etc/sysctl.d/99-sysctl.conf ...
* Applying /etc/sysctl.conf ...
```

Connectez-vous ensuite en tant que l'utilisateur **lxcnp** :

```
root@debian9:~# su - lxcnp
lxcnp@debian9:~$
```

Créez le répertoire **~/.config/lxc** :

```
lxcnp@debian9:~$ mkdir -p ~/.config/lxc
```

Créez trois entrées dans le fichier **~/.config/lxc/default.conf** afin d'y inclure la configuration du fichier **/etc/lxc/default.conf** et mapper l'UID et le GID :

```
lxcnp@debian9:~$ echo "lxc.include = /etc/lxc/default.conf" >> ~/.config/lxc/default.conf

lxcnp@debian9:~$ echo "lxc.id_map = u 0 165536 65536" >> ~/.config/lxc/default.conf

lxcnp@debian9:~$ echo "lxc.id_map = g 0 165536 65536" >> ~/.config/lxc/default.conf

lxcnp@debian9:~$ cat ~/.config/lxc/default.conf
lxc.include = /etc/lxc/default.conf
lxc.id_map = u 0 165536 65536
lxc.id_map = g 0 165536 65536
```

Déconnectez-vous et reconnectez-vous en tant que l'utilisateur **lxcnp** :

```
lxcnp@debian9:~$ exit
déconnexion
root@debian9:~# su - lxcnp
lxcnp@debian9:~$
```

Création du Conteneur

Créez maintenant un conteneur non-privilégié appelé **lxc-bb-np** à partir du gabarit **busybox** :

```
lxcnp@debian9:~$ lxc-create -n lxc-bb-np -t busybox
WARN: could not reopen tty: Permission denied
WARN: could not reopen tty: Permission denied
WARN: could not reopen tty: Permission denied
WARN: could not reopen tty: Permission denied
WARN: could not reopen tty: Permission denied
WARN: could not reopen tty: Permission denied
WARN: could not reopen tty: Permission denied
WARN: could not reopen tty: Permission denied
WARN: could not reopen tty: Permission denied
WARN: could not reopen tty: Permission denied
WARN: could not reopen tty: Permission denied
WARN: could not reopen tty: Permission denied
chmod: impossible d'accéder à '/home/lxcnp/.local/share/lxc/lxc-bb-np/rootfs/bin/passwd': Aucun fichier ou
dossier de ce type
setting root password to "root"
Failed to change root password

lxcnp@debian9:~$ ls -l /home/lxcnp/.local/share/lxc/lxc-bb-np/rootfs/
total 60
drwxr-xr-x 2 165536 165536 4096 juil. 29 13:11 bin
drwxr-xr-x 4 165536 165536 4096 juil. 29 13:11 dev
```

```
drwxr-xr-x 3 165536 165536 4096 juil. 29 13:11 etc
drwxr-xr-x 2 165536 165536 4096 juil. 29 13:11 home
drwxr-xr-x 2 165536 165536 4096 juil. 29 13:11 lib
drwxr-xr-x 2 165536 165536 4096 juil. 29 13:11 lib64
drwxr-xr-x 2 165536 165536 4096 juil. 29 13:11 mnt
drwxr-xr-x 2 165536 165536 4096 juil. 29 13:11 proc
drwxr-xr-x 2 165536 165536 4096 juil. 29 13:11 root
drwxr-xr-x 2 165536 165536 4096 juil. 29 13:11 sbin
drwxr-xr-x 2 165536 165536 4096 juil. 29 13:11 selinux
drwxr-xr-x 2 165536 165536 4096 juil. 29 13:11 sys
drwxr-xr-x 2 165536 165536 4096 juil. 29 13:11 tmp
drwxr-xr-x 7 165536 165536 4096 juil. 29 13:11 usr
drwxr-xr-x 3 165536 165536 4096 juil. 29 13:11 var
```

Contrôle du Mappage

Démarrez le conteneur :

```
lxcnp@debian9:~$ lxc-start -n lxc-bb-np
lxcnp@debian9:~$ lxc-ls -f --running
NAME      STATE   AUTOSTART GROUPS IPV4 IPV6
lxc-bb-np RUNNING 0         -      -    -
```

Attachez-vous au conteneur et contrôlez l'UID de l'utilisateur des processus dans le conteneur :

```
lxcnp@debian9:~$ lxc-attach -n lxc-bb-np
WARN: could not reopen tty: Permission denied
                                                WARN: could not reopen tty: Permission denied
          WARN: could not reopen tty: Permission denied


BusyBox v1.22.1 (Debian 1:1.22.0-19+b3) built-in shell (ash)
```

```
Enter 'help' for a list of built-in commands.

/ # ps aux
PID   USER     COMMAND
    1 root     init
    4 root     /bin/syslogd
    7 root     /bin/getty -L tty1 115200 vt100
    8 root     init
    9 root     /bin/sh
   10 root     ps aux
/ #
```

Détachez-vous du conteneur et sortez du compte **lxcnp** :

```
/ # exit
lxcnp@debian9:~$ exit
déconnexion
root@debian9:~#
```

Dernièrement, vérifiez l'UID de l'utilisateur des processus appartenant au conteneur :

```
root@debian9:~# ps aux | grep 165536
165536    3450  0.0  0.0   4832   256 ?        Ss   13:17   0:00 init
165536    3462  0.0  0.0   4836   100 ?        Ss   13:17   0:00 /bin/syslogd
165536    3465  0.0  0.0   4832   256 pts/0    Ss+  13:17   0:00 /bin/getty -L tty1 115200 vt100
165536    3466  0.0  0.0   4832   108 pts/2    Ss+  13:17   0:00 init
root      3485  0.0  0.0  12784   932 pts/0    S+   13:22   0:00 grep 165536
```

2.7 - Création d'un Conteneur Éphémère

Par défaut les conteneurs LXC sont permanents. Il est possible de créer un conteneur éphémère, c'est-à-dire un conteneur où toutes les données sont détruites à l'arrêt de celui-ci, en utilisant la commande **lxc-copy** ainsi que l'option de cette commande **–epheremal** ou **-e**.

La Commande lxc-copy

Notez que le conteneur d'origine doit être arrêté lors de l'utilisation de la commande **lxc-copy** :

```
root@debian9:~# lxc-ls -f --running
NAME   STATE   AUTOSTART GROUPS IPV4 IPV6
lxc-bb RUNNING 0         -      -    -

root@debian9:~# lxc-copy -e -N lxc-bb-eph -n lxc-bb
lxc-copy: lxccontainer.c: do_lxcapi_clone: 3079 error: Original container (lxc-bb) is running
```

Arrêtez donc le conteneur **lxc-bb** puis créez la copie :

```
root@debian9:~# lxc-stop -n lxc-bb
root@debian9:~# lxc-ls -f --running
root@debian9:~# lxc-copy -e -N lxc-bb-eph -n lxc-bb
Created lxc-bb-eph as clone of lxc-bb
```

Attachez-vous au conteneur **lxc-bb-eph** :

```
root@debian9:~# lxc-attach lxc-bb-eph
lxc-attach: missing container name, use --name option
root@debian9:~# lxc-attach -n lxc-bb-eph


BusyBox v1.22.1 (Debian 1:1.22.0-19+b3) built-in shell (ash)
Enter 'help' for a list of built-in commands.

~ #
```

Créez un fichier de contrôle appelé **testdata** :

```
~ # ls -l
```

```
total 4
-rw-r--r--    1 root     root            51 Jul 28 13:47 hi.sh
~ # pwd
/root
~ # echo "test" > testdata
~ # ls -l
total 8
-rw-r--r--    1 root     root            51 Jul 28 13:47 hi.sh
-rw-r--r--    1 root     root             5 Jul 29 15:14 testdata
~ #
```

Déconnectez-vous du conteneur puis attachez-vous de nouveau :

```
~ # exit

root@debian9:~# lxc-attach -n lxc-bb-eph


BusyBox v1.22.1 (Debian 1:1.22.0-19+b3) built-in shell (ash)
Enter 'help' for a list of built-in commands.

~ # ls -l
total 8
-rw-r--r--    1 root     root            51 Jul 28 13:47 hi.sh
-rw-r--r--    1 root     root             5 Jul 29 15:14 testdata
~ #
```

Important - Notez que le fichier **testdata** est toujours présent.

Déconnectez-vous de nouveau et arrêtez le conteneur :

```
~ # exit

root@debian9:~# lxc-stop -n lxc-bb-eph

root@debian9:~# lxc-ls
lxc-bb

root@debian9:~# lxc-start -n lxc-bb-eph
lxc-start: log.c: log_open: 300 failed to open log file "/var/lib/lxc/lxc-bb-eph/lxc-bb-eph.log" : No such file
or directory
lxc-start: tools/lxc_start.c: main: 317 Executing '/sbin/init' with no configuration file may crash the host
```

Important - Notez que le conteneur **lxc-bb-eph** a été détruit.

2.8 - Sauvegarde des Conteneurs

Un conteneur LXC peut être sauvegardé de trois façons différentes :

- utiliser la commande **tar** ou **cpio** pour créer un archive du répertoire **rootfs** et du fichier **config** associés au conteneur
- utiliser la commande **lxc-copy** sans l'option **-e**
- utiliser la commande **lxc-snapshot**

La Commande lxc-snapshot

Cette commande permet de gérer des instantanées des conteneurs. A noter que les conteneurs doivent être arrêtés avant de prendre une instantanée :

```
root@debian9:~# lxc-ls -f --running
```

```
root@debian9:~# lxc-snapshot -n lxc-bb
lxc-snapshot: lxccontainer.c: do_lxcapi_snapshot: 3407 Snapshot of directory-backed container requested.
lxc-snapshot: lxccontainer.c: do_lxcapi_snapshot: 3408 Making a copy-clone.  If you do want snapshots, then
lxc-snapshot: lxccontainer.c: do_lxcapi_snapshot: 3409 please create an aufs or overlayfs clone first, snapshot
that
lxc-snapshot: lxccontainer.c: do_lxcapi_snapshot: 3410 and keep the original container pristine.
```

Les snapshots sont stockés dans le sous-répertoire **snaps** du répertoire **/var/lib/lxc/<nom_conteneur>/**. Le premier s'appelle **snap0** :

```
root@debian9:~# ls -l /var/lib/lxc/lxc-bb
total 12
-rw-r--r--  1 root root 1102 juil. 28 13:04 config
-rw-r--r--  1 root root    0 juil. 28 13:14 lxc-bb.log
drwxr-xr-x 17 root root 4096 juil. 28 15:50 rootfs
drwxr-xr-x  3 root root 4096 juil. 29 17:34 snaps

root@debian9:~# ls -l /var/lib/lxc/lxc-bb/snaps/
total 4
drwxrwx--- 3 root root 4096 juil. 29 17:34 snap0

root@debian9:~# ls -l /var/lib/lxc/lxc-bb/snaps/snap0/
total 12
-rw-r--r--  1 root root 1110 juil. 29 17:34 config
drwxr-xr-x 17 root root 4096 juil. 28 15:50 rootfs
-rw-r--r--  1 root root   19 juil. 29 17:34 ts
```

L'horodatage de la création du snapshot est stocké dans le fichier **ts** :

```
root@debian9:~# cat /var/lib/lxc/lxc-bb/snaps/snap0/ts
2020:07:29 17:34:36root@debian9:~#
```

En comparant la taille du **rootfs** du conteneur d'origine ainsi que de son snapshot, on peut constater que les deux sont identiques :

```
root@debian9:~# du -sh /var/lib/lxc/lxc-bb/rootfs/
```

```
792K    /var/lib/lxc/lxc-bb/rootfs/

root@debian9:~# du -sh /var/lib/lxc/lxc-bb/snaps/snap0/rootfs/
792K    /var/lib/lxc/lxc-bb/snaps/snap0/rootfs/
```

Pour restaurer un conteneur identique à l'original, il convient d'utiliser de nouveau la commande **lxc-snapshot** :

```
root@debian9:~# lxc-snapshot -r snap0 -n lxc-bb -N lxc-bb-snap0

root@debian9:~# lxc-ls
lxc-bb       lxc-bb-snap0

root@debian9:~# lxc-start -n lxc-bb-snap0

root@debian9:~# lxc-attach -n lxc-bb-snap0


BusyBox v1.22.1 (Debian 1:1.22.0-19+b3) built-in shell (ash)
Enter 'help' for a list of built-in commands.

~ # exit
root@debian9:~#
```

Copyright © 2022 Hugh Norris.