Gestion de conteneurs en Bash

Les solutions de conteneurisation sont devenues de plus en plus complètes et sophistiquées, faisant suite à un succès auprès des développeurs et des administrateurs systèmes. Ces derniers ont été sensibles à de nombreux atouts comme la simplicité, la centralisation des images, la sécurité et la mutualisation des ressources qu'offre ce type de solution. Mais il n'est pas sans rappeler que ces solutions n'existent que par l'apport de fonctionnalités majeures au niveau du système d'exploitation. Nous allons voir que les développeurs Linux n'ont pas attendu ces solutions, pour proposer des logiciels en ligne de commandes permettant de contrôler finement les options du système et ainsi apporter des fonctionnalités proches de celles offertes par les stars du domaine.

1. Btrfs

Afin de simuler au mieux notre gestionnaire de conteneurs, nous allons nous appuyer sur un système de fichiers nous permettant de réaliser des instantanés facilement. En effet, ext3/ext4 ne permettent pas de faire ça sans utiliser **LVM** comme gestionnaire de volume. Des solutions telles que **Docker** reposent sur une gestion des images de système de fichiers en mode *Copy On Write*, c'est-à-dire qu'elles vont réaliser une copie seulement dans le cas d'une écriture. Docker va donc ensuite gérer des couches, pouvant s'apparenter à des instantanés, qu'il va fusionner dans une couche placée tout en haut de la pile (elle est la seule couche accessible en écriture), et qui va représenter la zone de travail du conteneur.

Btrfs (*B-Tree file system*) est un candidat idéal, car c'est est un système de fichiers en mode *copy on write*, au contraire de systèmes ext qui sont eux basés sur de la journalisation. Btrfs possède donc l'avantage d'avoir des outils de gestion d'instantanés, lui conférant un atout indéniable pour de la gestion de conteneurs.

Nous allons tout d'abord installer Btrfs :

$ sudo apt install btrfs-tools

Il va falloir ensuite initier notre système de fichiers ; pour ce faire, nous devons posséder une partition btrfs. Nous allons donc utiliser un disque que nous allons formater :

mkfs.btrfs /dev/sd[bcde]

Puis nous allons monter ce dernier dans le répertoire **/var/work** :

mkdir /var/images

mount /dev/sd[bcde] /var/work

Une fois notre point de montage réalisé, il est facile de créer de nouveau sous volume via la commande suivante :

btrfs suvolume create /var/work/jessie

Une fois notre volume créé, nous pouvons y placer une image de type Docker, ou encore pour les utilisateurs **Debian**, via l'utilisation de l'utilitaire **debootstrap**, de télécharger un nouvel **rootfs** d'une version précise de votre Debian préféré :

deboostrap jessie /var/work/jessie

Ensuite, la puissance du Btrfs entre en jeu. Nous pouvons lister les volumes et les instantanés d'un point de montage via la commande suivante :

btrfs subvolume list /var/work

Pour simuler la gestion des commits de nos images, nous allons utiliser le gestionnaire d'instantané du btrfs via la commande :

btfs subvolume snapshot /var/work/jessie /var/work/jessie.1

Docker s'appuie essentiellement sur le **aufs**, faisant partie de la famille des **unionfs**. Ces derniers ont pour vocation de fusionner les points de montage de différents types de systèmes de fichiers. Ici Btrfs est lui un

système de fichiers, dont l'écosystème fournit un grand nombre d'utilitaires en ligne de commandes pour la gestion de volumes et instantanés.

2. Cgroups

Les *cgroups*, faisant partie d'un binôme indissociable avec les *namespaces*, sont une fonctionnalité du noyau permettant de limiter les ressources du système (mémoire, cpu, etc.) pour un ensemble de processus faisant partie d'un groupe défini. Toutes les solutions de conteneurisation s'appuient sur cette fonctionnalité pour gérer les ressources utilisées par chaque conteneur.

Les *cgroups* utilisent des concepts avancés du système, mais les développeurs Linux ont fourni des utilitaires en ligne de commandes permettant de manipuler et configurer simplement ces derniers.

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

Nous allons ensuite créer un *cgroup* auquel nous allons attacher trois sous-systèmes :

- **cpu** pour la gestion de la consommation du processeur ;
- **cpuactt** (*cpu accounting*) pour avoir une gestion plus fine sur la consommation de chaque conteneur ;
- **memory** pour la gestion de la mémoire vive et le swapping.

```
# cgcreate -g cpu,cpuactt,memory:/jessie
```

Nous allons ensuite attribuer à ces sous-systèmes des valeurs limites pour nos futurs conteneurs via la commande **cgset** :

```
# cgset -r cpu.shares=200 "jessie"
```

```
# cgset -r memory.limit_in_bytes="100000" "jessie"
```

Ici nous avons attribué deux fois plus de temps processeur et seulement **100000** octets de mémoire à tous les processus faisant partie du groupe

jessie.

Enfin, via la commande **cgexec**, nous pouvons exécuter un processus dans le groupe jessie :

$ cgexec -g cpu,cpuactt,memory:/jessie /bin/bash

Maintenant que nous savons gérer nos ressources système, nous allons cloisonner nos conteneurs, afin de nous prévenir d'interactions non maîtrisées entre nos processus.

3. Namespace

La gestion des *namespaces* est la fonctionnalité majeure de ces dix dernières années dans le noyau Linux. Sans eux, il n'existerait pas de conteneurs à proprement parlé. Les *namespaces* représentent l'évolution vers ce que l'on appelle « la virtualisation légère ». Ils permettent de gérer plusieurs « Systèmes d'exploitation » s'exécutant sur le même noyau. Ceci permet de faire de la mutualisation à moindre coup. Combinées avec les *cgroups*, ces deux technologies sont les pierres angulaires des techniques de conteneurisation.

Les *namespaces* permettent de cloisonner les processus selon cinq critères :

- **mount** pour les points de montages ;
- **IPC** : *Inter Process Call* ;
- **UTS** : *hostname* et nom de domaine NIS ;
- **network** : réseau interface ;
- **PID** : espace de *process* id ;
- **User** pour les utilisateurs.

Pour le sujet de cet article, nous allons créer un nouveau *namespace* pour chaque conteneur. Seul le *namespace* réseau va nécessiter une gestion un peu plus fine, afin d'offrir à chaque conteneur un réel réseau cloisonné.

Pour gérer les *namespaces*, les développeurs du noyau Linux proposent

deux outils en ligne de commandes, essentiellement calqués sur les interfaces bas niveau des API *namespaces* : **unshare** et **ip netns**.

Le premier permet de manipuler tous les types de *namespaces*, et le dernier permet de manipuler plus finement les *namespaces* réseau. Ces deux utilitaires sont présents par défaut sur des distributions telles qu'**Ubuntu** ou Debian.

Afin de lancer un Bash dans un nouveau *namespace*, nous allons donc utiliser **unshare**. **unshare** permet de se « détacher » de son *namespace* parent en en créant un nouveau :

```
# unshare -fpium --mount-proc /bin/bash
```

Le processus **bash** va donc tourner dans ce que l'on peut appeler un « cloisonnement du système », ou encore un « conteneur ». Seule, la commande **unshare** permet de réaliser 90 % du travail et elle ne fait en aucun cas partie d'un logiciel de conteneurisation. **unshare** permet aussi de créer un *namespace* réseau, mais nous allons voir que **ip**, l'utilitaire de base de gestion des interfaces réseau, incorpore toute une gestion des *namespaces*.

La gestion du réseau est un peu plus complexe, mais pas irréalisable. La sécurité des *namespaces* impose qu'une interface ne peut appartenir qu'à un seul *namespace*. Donc toutes les interfaces physiques appartiennent au *namespace* réseau racine et ne peuvent donc pas être transférées dans un autre *namespace*.

Nous allons donc utiliser la même technique que dans la plupart des solutions de conteneurisation, c'est-à-dire, un duo d'interface virtuelle combiné avec un *bridge*.

Nous allons tout d'abord créer le duo d'interface virtuelle, qui va nous permettre d'avoir une patte dans chaque *namespace* :

```
# ip link add dev veth0 type veth peer name veth1
```

Ensuite, nous allons associer à la première interface le *bridge*, qu'il faut au préalable créer, configurer et sécuriser ; mais ceci ne fait pas l'objet de cet article :

```
# ip link set dev veth0 up

# ip link set veth0 master bridge
```

Puis, nous allons créer notre *namespace* réseau. **ip** propose donc une option **netns**, permettant de réaliser les mêmes opérations de façon classique tout en précisant un *namespace* :

Puis, nous allons associer notre nœud **veth1** au nouveau *namespace* créé :

```
# ip link set veth1 netns jessie
```

Enfin, nous pouvons réaliser sur ce dernier, dans son *namespace*, une configuration pour l'ensemble de notre conteneur :

```
# ip netns exec jessie ip addr add 10.0.0.2/24 dev veth1

# ip netns exec jessie ip link set dev veth1 up

# ip netns exec jessie ip route add default via 10.0.0.1
```

Ensuite, nous pouvons lancer un processus **bash** dans ce nouveau *namespace* :

```
# ip netns exec jessie /bin/bash
```

Enfin, une fois notre conteneur fini, nous pouvons le supprimer via la commande :

4. Cash (Container in Bash)

Nous allons faire un parallèle entre les fonctionnalités communes des gestionnaires de conteneurs, et ce qui est possible avec les outils disponibles. Nous supposons que nous avons un point de montage btrfs configuré en **/var/work**.

Nous pouvons donc simuler la fonction **pull** de la manière suivante :

function cash_pull**()** {

btrfs subvolume create "/var/work/$1"

debootstrap "$2" "/var/work/$1"

}

Ici nous créons un nouveau sous-volume **btrfs**, pour y écrire une distribution de notre choix :

Ensuite, nous pouvons simuler l'exécution d'un nouveau conteneur avec la fonction suivante :

function cash_run**()** {

ip **link** add dev veth0_"$1" **type** veth peer name veth1_"$1"

ip **link set** dev veth0_"$1" up

ip **link set** veth0 master bridge

ip netns add "$1"

ip **link set** veth1 netns "$1"

ip netns **exec** "$1" ip addr add 10.0.0."$2"/24 dev veth1_"$1"

ip netns **exec** "$1" ip **link set** dev veth1_"$1" up

ip netns **exec** "$1" ip route add default via 10.0.0.1

cgcreate -**g** "cpu,cpuacct,memory:/$1"

cgset -r cpu.shares="$3" "$1"

cgset -r memory.limit_in_bytes="$4" "$1"

```
cgexec -g "cpu,cpuacct,memory:$1" \

ip netns exec "$1" \

unshare -fmuip --mount-proc \

chroot "/var/work/$1" \

${@:5}

ip link del dev veth0_"$1"

ip netns del "$1"

}
```

La première partie est similaire à ce que nous avons vu dans la section *namespace* réseau. Ensuite, nous configurons les *cgroups*, et enfin nous exécutons notre commande dans l'ensemble des *namespaces* créés, via les commandes **cgexec** pour exécuter dans un *cgroup*, **ip netns** pour utiliser le *namespace* réseau précédemment créé, **unshare** pour tous les autres *namespaces*, et **chroot** pour définir la racine de notre nouveau *namespace* dans le système que nous avons tiré lors du **cash_pull**.

```
# cash_run my 2 100 1000000 /bin/bash
```

Enfin, nous pouvons simuler le commit via les fonctionnalités de *snapshot* du système btrfs.

```
function cash_commit() {

btrfs subvolume snapshoot "/var/work/$1" "/var/work/$1.$2"

}
```

Que l'on appelle avec :

En améliorant un peu la gestion des fichiers et instantanés, nous pouvons avoir un résultat vraiment proche des logiciels du marché.

Conclusion

Le succès des gestionnaires de conteneurs repose essentiellement sur des fonctionnalités du système. Mais nous pouvons constater qu'il est aisé d'accéder à des notions avancées via des outils simples, et qu'en les combinant bien nous réalisons des choses réellement puissantes.