C'EST QUOI EXACTEMENT UN CONTENEUR ?

Introduction

Depuis le début de ce cours on parle de conteneur, mais on parle de qui/quoi exactement ? Qu'est-ce que c'est un conteneur ? Comment fonctionne-t-il ? À quoi servent-ils ? Je vais tenter de répondre à toutes ces questions à travers cet article.

Avant toute chose, il faut savoir que le noyau Linux offre quelques fonctionnalités comme les **namespaces** (ce qu'un processus peut voir) et les **cgroups** (ce qu'un processus peut utiliser en terme de ressources), qui vont vous permettre d'**isoler les processus Linux** les uns des autres. Lorsque vous utilisez ces fonctionnalités, on appelle cela **des conteneurs**.

Prenons un exemple simple, si jamais on souhaite créer un conteneur contenant la distribution Ubuntu. Fondamentalement, ces fonctionnalités d'isolation proposées par le noyau Linux, vont vous permettent de prétendre d'avoir quelque chose qui ressemble à une machine virtuelle avec l'OS Ubuntu, sauf qu'en réalité ce n'est pas du tout une machine virtuelle mais un **processus isolé** s'exécutant dans le même noyau Linux .

> **Information**
>
> Docker tire parti de plusieurs ces fonctionnalités proposées par le noyau Linux pour fournir ses fonctionnalités.

Voyons voir plus en détail ces fonctionnalités.

Les namespaces : limiter les vues

Supposons que nous voulons créer une sorte de machine virtuelle. Une des caractéristiques que vous exigerez sera la suivante : "mes processus doivent être séparés des autres processus de l'ordinateur"

Pour réussir à atteindre notre but, on utilisera une fonctionnalité que Linux fournit à savoir les namespaces !

Les namespaces (ou "espaces de noms" en français) isolent les ressources partagées. Ils donnent à chaque processus sa propre vue unique du système, limitant ainsi leur accès aux ressources système sans que le processus en cours ne soit au courant des limitations.

Il éxiste différents types de namespaces, que je vais vous expliquer sur la liste ci-dessous :

- Le **namespace PID** : fournit un ensemble indépendant d'identifiants de processus (PID). Il s'agit d'une structure hiérarchique dans laquelle le namespace parent peut afficher tous les PID des namespaces enfants. Lorsqu'un nouveau namespace est créé, le premier processus obtient le PID 1 et constitue une sorte de processus init de ce namespace. Cela signifie également que si on tue ce processus PID 1 alors on mettra immédiatement fin à tous les processus de son namespace PID et à tous ses descendants.
- Le **namespace IPC** : empêche la communication avec les autres processus, plus simplement il interdit l'échange d'informations avec les autres processus.

- Le **namespace NET** : crée une pile réseau entièrement nouvelle, y compris : un ensemble privé d'adresses IP, sa propre table de routage, liste de socket, table de suivi des connexions, pare-feu et autres ressources liées au réseau.
- Le **namespace MOUNT** : monte un système de fichier propre au processus qui est différent du système de fichier de la machine hôte. Vous pouvez ainsi monter et démonter des systèmes de fichiers sans que cela n'affecte le système de fichiers hôte.
- Le **namespace USER** : fournit à la fois l'isolation des privilèges et la séparation des identifications d'utilisateurs entre plusieurs ensembles. Il permet par exemple de donner un accès root dans le conteneur sans qu'il soit root sur la machine hôte.
- Le **namespace UTS** : associe un nom d'hôte et de domaine au processus pour avoir son propre hostname.

Ce n'est pas vraiment le but de ce cours mais pour s'amuser un peu, on utilisera la commande **unshare** pour créer un namespace PID du programme bash.

Juste avant de lancer la commande **unshare**, je vais vous montrer les processus qui tournent déjà sur ma machine hôte :

```
ps aux
```

Résultat :

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   8324   156 ?        Ss   09:18   0:00 /init
root         3  0.0  0.0   8328   156 tty1     Ss   09:18   0:00 /init
hatim        4  0.0  0.0  16796  3424 tty1     S    09:18   0:00 -bash
root        16  0.0  0.0   8332   160 tty2     Ss   10:08   0:00 /init
hatim       17  0.0  0.0  16788  3420 tty2     S    10:08   0:00 -bash
```

```
root         75  1.8  0.1 225388 16196 ?        Ss   11:24   0:00 /usr/sbin/apache2 -
www-data     80  0.0  0.0 225680  2796 ?        S    11:24   0:00 /usr/sbin/apache2 -
www-data     81  0.0  0.0 225680  2796 ?        S    11:24   0:00 /usr/sbin/apache2 -
www-data     82  0.0  0.0 225680  2796 ?        S    11:24   0:00 /usr/sbin/apache2 -
www-data     83  0.0  0.0 225680  2796 ?        S    11:24   0:00 /usr/sbin/apache2 -
www-data     84  0.0  0.0 225680  2796 ?        S    11:24   0:00 /usr/sbin/apache2 -
mysql       130  2.0  0.0  10660   800 ?        S    11:24   0:00 /bin/sh /usr/bin/mys
mysql       493  8.4  1.0 1934168 129464 ?      Sl   11:24   0:00 /usr/sbin/mysqld --
hatim       551  0.0  0.0  17380  1924 tty2     R    11:24   0:00 ps aux
```

Maintenant exécutant notre namespace PID avec la commande unshare :

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

Je vais maintenant afficher les processus en cours au sein de ce mini conteneur :

```
ps aux
```

Résultat :

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.5  0.0  16692  3292 tty2     S    11:27   0:00 bash
root         9  0.0  0.0  17380  1920 tty2     R    11:27   0:00 ps aux
```

Il n'y a que 2 processus en cours d'exécution bash et ps. Preuve que les namespaces permettent de limiter la vue d'un processus.

En tout bravo vous venez de créer un tout mini conteneur .

Tapez ensuite la commande `exit` pour quitter votre mini conteneur.

Les Cgroups : limiter les ressources

Bon jusqu'ici, nous avons vu comment fonctionnent les namespaces, mais que faire si je veux limiter la quantité de mémoire ou de processeur utilisée par l'un de mes processus ? Heureusement que des personnes en 2007 ont développé

spécialement pour nous les **groupes de contrôle**.

.

> **Information**
>
> Il éxiste aussi l'outil nommé nice (et son petit frère renice) permettant de contrôler la priorité des processus sur Linux, sauf que les groupes de contrôle proposent plus de fonctionnalités.

Je vous présente ci-dessous quelques types de cgroup :

- **cgroup cpuset** : assigne des processeurs individuels et des nœuds de mémoire à des groupes de contrôle
- **cgroup cpu** : planifie un accès aux ressources du processeur
- **cgroup cpuacct** : génère des rapports sur les ressources du processeur utilisées
- **cgroup devices** : autorise ou refuse l'accès aux périphériques
- **cgroup net_prio** : permet de définir la priorité du trafic réseau
- **cgroup memory** : définit la limite d'utilisation de la mémoire
- **cgroup blkio** : limite de la vitesse E/S (lecture/écriture) sur périphériques de type bloc (ex : disque dur)
- **cgroup pid** : limite le nombre de processus

Allé pour s'amuser encore un peu plus, créons un cgroup qui limite la mémoire !

Commençons d'abord par créer un cgroup limitant l'utilisation de la mémoire :

```
sudo cgcreate -a <nom_d_utilisateur> -g memory:<nom_du_cgroup>
```

Voyons ce qu'il y a dedans :

```
ls -l /sys/fs/cgroup/memory/<nom_du_cgroup>/
```

Résultat :

```
-rw-r--r-- 1 <nom_d_utilisateur> root 0 Okt 10 23:16 memory.kmem.limit_in_bytes
-rw-r--r-- 1 <nom_d_utilisateur> root 0 Okt 10 23:14 memory.kmem.max_usage_in_bytes
```

Ensuite, on va limiter notre cgroup à 20 mégaoctets :

```
sudo echo 20000000 >  /sys/fs/cgroup/memory/<nom_du_cgroup>/memory.kmem.limit_in_byte
```

Maintenant utilisons notre cgroup sur notre programme bash :

```
sudo cgexec  -g memory:<nom_du_cgroup> bash
```

Voilà le processus bash ne peut plus dépasser 20 Mo de mémoire.

Conclusion

Pour résumer, la technologie Docker possède de nombreuses fonctionnalités de nos jours, mais beaucoup d'entre elles reposent sur les fonctionnalités de base du noyau Linux vues plus haut.

Pour rentrer plus dans les détails, les conteneurs contiennent généralement un ou plusieurs programme(s) de manière à les maintenir isolées du système hôte sur lequel elles s'exécutent. Ils permettent à un développeur de conditionner une application avec toutes ses dépendances, et de l'expédier dans un package unique.

En outre, ils sont conçus pour faciliter la mise en place d'une expérience cohérente lorsque les développeurs et les administrateurs système déplacent le code des

environnements de développement vers la production de manière rapide et reproductible.