R320 : Containers Docker

Guillaume Urvoy-Keller

1er novembre 2022

Vous devez faire un compte-rendu électronique à envoyer à Anderson.LOURENCO-DE-ARAUJO@univ-cotedazur.fr en fin de séance

1 On s'échauffe un peu sur les namespaces en Linux et OvS

Les namespaces sous Linux permettent de créer des processus qui se partagent ou non des espaces réseaux, des espaces de nommage (PID des processus), ... Il y a 6 namespaces actuellement :

1. Point de montage (le premier sur la planète namespace)
2. NS 2. Nom de host (hostname) – UTS
3. Processus (init virtuel) – PID
4. Communication inter-processus – IPC
5. Représentation des utilisateurs – USER
6. Réseaux – NET

1.1 Namespace UTS

La commande unshare permet de créer des processus qui partagent certaines caractéristiques avec le processus père. Un exemple avec le namespace UTS :

1. Créez une machine virtuelle Debian 64 bits avec le script createvm et démarrez-là au travers de l'interface Virtualbox.
2. Logez-vous dans la machine en root.
3. Ouvrir un terminal puis tapez **uname -n** pour avoir le nom de la machine.
4. Ouvrir une seconde fenêtre dans le terminal (ctrl+shit+t) puis démarrer un nouveau bash qui ne partage pas le namespace UTS : **unshare -u /bin/bash** (pour créer un processus, il faut être root comme vous êtes en train de vous en rendre compte). Exécutez ensuite dans ce bash **hostname windows** puis faites un **uname -n** dans les 2 fenêtres.
5. Analysons les namespaces de chaque processus (premier et deuxième terminal). Pour cela, il faut :
   (a) Récupérer le numéro du processus courant avec **echo $$**
   (b) ..puis faire un **ls -al /proc/numéro processus/ns** . Les numéros des namespaces sont entre []. Qu'est-ce qui diffère et qu'est-ce qui est commun entre les 2 processus ?

1.2 Namespace Réseau

1. On recommence avec nos 2 fenêtres de terminaux. Sur la seconde, on crée un bash qui ne partage pas la partie réseau avec son pair : **unshare -n /bin/bash**
2. tapez un **ip a s** dans chaque terminal pour voir les interfaces.
3. On va relier les 2 namespaces réseaux en commençant par créer l'equivalent d'un fil ethernet dans le système d'exploitation. On appelle cela une paire ethernet virtuelle (veth en abrégée). On la crée, dans la première fenêtre (namespace par défaut), avec la commande **sudo ip link add type veth**. Vous devriez la voir avec **ip a s** sous la forme de 2 interfaces veth0@veth1 et veth1@veth0.

4. Comme avec un fil ethernet, on va en prendre un bout et le faire changer de namespace avec la commande ip : **sudo ip link set veth1 netns numéro_processus_second_namespace**
5. Un petit **ip a s** dans le second namespace devrait vous dire que l'interface est là... et qu'elle n'est plus dans le premier car une interface est dans un seul namespace !
6. On va ensuite mettre les 2 interfaces en up avec la commande **ip link set dev veth1 up** et la même chose pour l'autre veth0 dans l'autre namespace.
7. On leur assigne des adresses dans le réseau 10.0.0.0/24 et on montre que tout cela fonctionne (ping).

2 Configuration hôte Docker et premiers containers

1. Pour installer docker, il faut ajouter des dépôts spécifiques de paquets pour Debian. A priori, il va falloir exécuter les commandes suivantes, que vous pouvez copier-coller depuis la page `https://docs.docker.com/engine/installation/linux/debian/` :
2. Puisque l'IUT utilise un proxy, il faut l'indiquer à Docker. Pour cela, il faut créer le répertoire suivant :

   ```
   mkdir /etc/systemd/system/docker.service.d
   ```

   puis ajoutez le fichier http-proxy.conf dans ce répertoire et écrire dedans :

   ```
   [Service]
   Environment="HTTP_PROXY=http://wall.unice.fr:3128/"
   ```

   Il faut ensuite redémarrer docker :

   ```
   systemctl daemon-reload
   systemctl restart docker
   ```

   Vérifiez que Docker est bien installé en faisant l'équivalent d'un hello-world que l'on fait en apprenant tout nouveau language de programmation : **sudo docker run hello-world**. Docker liste les containers locaux (vous le ferez vous-même sous peu), découvre qu'il n'a pas le container hello-world et va le télécharger depuis le dépôt par défaut (le hub docker). Il va ensuite démarrer le container et l'exécuter. Vous devriez donc obtenir globalement quelque chose d'équivalent à :

   ```
   sudo docker run hello-world
   Unable to find image 'hello-world:latest' locally
   latest: Pulling from library/hello-world
   b901d36b6f2f: Pull complete
   0a6ba66e537a: Pull complete
   Digest: sha256:....
   Status: Downloaded newer image for hello-world:latest

   Hello from Docker.
   This message shows that your installation appears to be working correctly.
   [...]
   ```

3. Nous allons maintenant télécharger un container plus intéressant que hello-world : le container officiel ubuntu. Ce container est **officiel** car c'est l'équipe de la distribution ubuntu.
   (a) Naviguez sur le hub docker `https://hub.docker.com/` **en cliquant sur Explore en haut à gauche** et trouvez les images officielles d'ubuntu, debian, mysql, apache, etc. Une image est officielle si elle est annotée comme telle.Vous pouvez ne sélectionner que les images officielles à l'aide du bouton All qui offre cette option.
   (b) Installez sur votre machine Debian la dernière version de l'image ubuntu : **docker pull ubuntu**.
   (c) Vérifiez que l'image est correctement installée en tapant **docker images**. Quand est-ce que l'image a été créée (par les gens d'Ubuntu) ?

3 Premier container

1. Démarrons notre premier containers ubuntu. **docker run ubuntu**. Que se passe-t-il ?
2. Un container ne reste en vie que si un processus est actif. On peut lister les containers actifs avec la commande **docker ps**. On peut aussi lister tous les containers, actifs ou inactifs avec **docker ps -a**. Que vous retourne ces commandes et pourquoi ?
3. Nous allons maintenant rediriger l'entrée standard du container avec l'option -i et ouvrir un pseudo-terminal avec -t, le tout en exécutant le processus /bin/bash :

   ```
   docker run -ti --name=ubuntu ubuntu /bin/bash
   ```

   Quelle est la version d'ubuntu du container (tapez **lsb_release -a** qui est dans le *package* lsb-release) ?
4. Ouvrez un second terminal. Listez les containers actifs ? Combien y en a-t-il ?
5. Le principe d'un container est qu'il ressemble à une VM vu de l'intérieur, mais qu'il est en ensemble de processus vus de l'extérieur. L'extérieur ici, c'est la machine hôte Debian. Placez vous dans votre second terminal (celui de l'hôte).
   (a) Listez les processus avec top
   (b) Ajoutez le namespace des processus afin de voir le processus auquel en tapant **f** dans le top puis en allant sélectionner le champ **nsPID**. Une fois ce champ sélectionner avec les flèches et la barre d'espace, retournez à l'affichage général en tapant q
   (c) top liste les processus par défaut par leur activité CPU. On va donc "faire du bruit" dans le container avec la commande **yes**.
   (d) Alternativement, on peut classer les processus suivant le critère désiré. Nous allons les classer de façon décroissante en tapant à nouveau **f** puis en se plaçant sur la a colonne nsPID et en tapant *s*.
   (e) Quel est le namespace id de votre container ? Vérifiez que vous voyez tous les processus de votre container en listant également dans le container avec la commande top.
6. Sortez de votre container et tuez-le :

   ```
   docker rm ubuntu
   ```

7. Recréer le même container que précédemment :

   ```
   docker run -ti --name=ubuntu ubuntu /bin/bash
   ```

8. Un container obtient des interfaces réseaux qui sont privées, et séparées de la machine hôte.
   (a) Listez les interfaces réseaux de votre container.
   (b) Quelle est l'adresse IP de votre container ? La commande ip est dans le *package* iproute2
   (c) Docker assigne une adresse privée à chaque container et lui donne accès au reste de l'internet au travers d'une passerelle qui est l'interface virtuelle docker de l'hôte. Pour bien le comprendre
      i. Trouvez la passerelle par défaut de votre container avec la commande **route** ou **netstat -rn**. La commande netstat est dans le *package* net-tools.
      ii. Vérifiez que c'est bien une des interfaces de la machine hôte Debian (quel est son nom ?).
9. Nous allons maintenant arrêter notre container.
   (a) .. c'est pas bien compliqué, il suffit de faire un **exit** dans le container ou un **docker stop ubuntu** depuis l'hôte debian.
   (b) ...mais vous pouvez le redémarrer en tapant : **docker start ubuntu**. Vérifiez que le container est actif.
   (c) Le problème est qu'il faut se connecter au container (puisqu'il s'agit ici d'une machine virtuelle légère). Il suffit de le faire avec la commande **docker attach ubuntu** (parfois il faut appuyer sur une touche du clavier pour avoir le prompt). Testez avec votre container.
10. Nous allons maintenant tuer notre container. Cela se fait avec la commande **docker rm ubuntu**. Cela n'est possible que si le container est arrêté !

4 Gestion des containers

1. Nous allons redémarrer un autre container basé sur l'image ubuntu avec un nom différent pour changer :
   `docker run -ti --name=bob ubuntu /bin/bash`
2. On peut inspecter ce qui se passe dans le container depuis la machine hôte avec la commande **docker logs bob** ou mieux encore **docker logs -f bob** qui est (un peu) l'équivalent du **tail -f /var/log/syslog** que l'on ferait sur la machine hôte. On peut d'ailleurs exécuter cette commande pour voir le démon docker qui reporte les manipulations que vous effectuées (démarrage, arrêt de containers, etc.). Faites le test en faisant un **ps** dans le container et en ayant le logs ouvert en continue depuis l'hôte. Puis faites un **top** dans le container.
3. Si on veut surveiller efficacement un container, il faut non seulement lire ses logs, mais également avoir une estampille temporelle avec l'option t : **docker logs -ft bob**
4. Docker fournit aussi des statistiques sur l'usage des resources CPU/mémoire/réseau des containers : **docker stats bob**. Pour que cela soit intéressant, il faut que le container soit actif. Lancez la commande **yes** et vérifiez les statistiques.

5 Gestion des images

1. Commençons par lister les images disponibles : **docker images**.
2. Pour ajouter une image localement, il suffit de faire un simple **docker pull xxx**. Allez sur le hub docker et cherchez une image Apache officielle (aussi connu sous le nom **httpd**) puis téléchargez la.
3. Nous allons maintenant créer notre propre image qui va nous permettre de lancer un serveur Web apache. Pour cela, il faut définir la méthode de construction du container dans un fichier.
   (a) Créez un répertoire
   `~/Docker/Apache`
   (b) Dans ce répertoire, créez un fichier Dockerfile. Dans ce fichier, mettez les commandes :

```
FROM ubuntu:latest
   MAINTAINER Votre nom
   RUN apt-get -yqq update && apt-get install -yqq apache2

   WORKDIR /var/www/html

   ENV APACHE_RUN_USER www-data
   ENV APACHE_RUN_GROUP www-data
   ENV APACHE_LOG_DIR /var/log/apache2
   ENV APACHE_PID_FILE /var/run/apache2.pid
   ENV APACHE_RUN_DIR /var/run/apache2
   ENV APACHE_LOCK_DIR /var/lock/apache2

   RUN mkdir -p $APACHE_RUN_DIR $APACHE_LOCK_DIR $APACHE_LOG_DIR

   ENTRYPOINT [ "/usr/sbin/apache2" ]
   CMD ["-D", "FOREGROUND"]
   EXPOSE 80
```

   (c) Créons l'image **sudo docker build -t="votre_nom/apache" .** Attention à ne pas le oublier le point '.' à la fin de la commande (qui indique que le fichier Dockerfile est dans le répertoire local. Le nom du container est nécessairement du votre_nom/apache et non apache car apache serait interprété comme un répertoire officiel
   (d) Listez vos images : **docker images**.

6 Faire tourner des applications avec Docker

1. Jusqu'à présent, nous avons fait tourner des containers ubuntu en mode interactif. C'est intéressant, mais dans un contexte de production où un opérateur veut démarrer beaucoup de containers sur une machine, on veut démarrer les containers en mode démon, c'est-à-dire en tâche de fond. Cela se fait simplement avec l'option -d.
   (a) Commençons par un simple **docker run -d --name=demon ubuntu /bin/bash**. Que se passe-t-il ? Faites un **docker ps** et **docker ps -a** pour comprendre
   (b) Détruisez le container précédent. Nous allons corriger le problème précédent en faisant quelque chose dans le container :

   ```
   docker run -d --name=demon ubuntu /bin/sh -c "while true; do echo hello
       world; sleep 1; done"
   ```

   (c) Montrez ce qui se passe dans le container depuis la machine hôte puis détruisez le container.
2. Démarrez un container votre_nom/apache en mode démon en exposant le port 80 : **docker run -d -p 80 --name=apache votre_nom/apache**
3. La dernière commande du fichier Dockerfile précédent permet d'exposer un port interne du container (le port 80 du serveur apache) depuis l'hôte debian. Pour trouver quel es tle port choisi par Docker, il suffit de taper **docker port apache 80**. Faites un test en ouvrant votre navigateur et vous connectant sur la machine locale (l'hôte debian) sur le port docker.
4. On peut mieux controller le port choisi. Si par exemple on veut que cela soit le port 80, il suffit de modifier la commande précédente :

   ```
   docker run -d -p 80:80 --name=apache votre_nom/apache
   ```

   Stoppez, détruisez et recréez votre container pour vérifier que cela fonctionne.
5. On veut maintenant en plus contrôler le contenu du serveur Web depuis l'hôte debian. Pour cela on va :
   (a) créer un répertoire **mkdir website** dans le répertoire Apache créé précédemment.
   (b) mettre dans ce répertoire un fichier index.html avec le contenu suivant :

   ```
   <html>
   Hello world
   </html>
   ```

   (c) démarrer votre container en montant le répertoire apache là où le serveur Apache dans le container va chercher ses données :

   ```
   docker  run -d -p 80:80  -v ~/Docker/Apache/website:/var/www/html
   --name=apache votre_nom/apache
   ```

   (d) vérifiez que le container offre le service attendu.

7 Docker *Swarm*

Docker *Swarm* permet de transformer plusieurs serveurs faisant du docker en un *pool* de machines sur lesquels on va exécuter des services (ensemble de containers).

1. Faites le ménage sur la VM en tuant tous les containers. Il faut le faire en 2 temps : 1) on arrête, 2) on tue :

   ```
   docker stop $(docker ps -aq)
   docker rm $(docker ps -aq)
   ```

2. Commencez par cloner votre machine virtuelle. Modifiez l'adresse MAC car le DHCP raisonne sur l'adresse MAC en salle de TP. Démarrez la seconde machine et vérifiez que vous avez bien la connectivité réseau avec des adresses IP distinctes pour les 2 VMs.
3. Sur la première VM, initiez le *swarm* avec la commande

```
docker swarm init --advertise-addr IP_VM
```

où IP_VM est l'adresse IP de **votre première** VM en 10.4.110.x ou 104.4.105.x ou 10.4.102.x.

4. Recopiez la sortie de la commande précédente , qui doit être du type

```
docker swarm join --token SWMTKN-1-2
    hdpt85zfkdnbbbwagfunjifx14njv4rg9uwahdjdgt8ukgemb-
  djtq1xm72dqfezpkf68vnkk2j IP:2377
```

dans la seconde VM.

5. Vérifiez les noeuds (VMs) du *swarm* avec la commande

```
docker node ls
```

6. Créer un premier service :

```
docker service create --name service1 --mode global alpine ping kheops.unice.
    fr
```

7. Vérifiez les containers créés sur chaque VM avec

```
docker ps
```

8. On peut aussi vérifiez au niveau service ce qui se passe :

```
docker service ps service1
```

9. Tuez votre service :

```
docker service rm service1
```

10. Rédémarrez le dans le mode répliqué et non le mode global :

```
docker service create --name service1 alpine ping kheops.unice.fr
```

11. Combien voit-on de containers sur les VMs ?
12. Que fait la commande :

```
docker service update service1  --replicas 6
```

13. On va maintenant lancer un service à partir de l'image contenant le serveur Apache que vous avez créer précédemment. Adaptez pour cela la commande ci-dessous à votre cas et montrez le résultat obtenu :

```
docker service create --name search --publish 9200:9200 --replicas 2
    elasticsearch
```

Qu'est-ce qui a été installé ?

14. Connectez-vous avec le navigateur de la machine physique sur les IPs des 2 VMs. Que voyez-vous ?
15. Faites un *down scaling* à 1 du service apache.
16. Connectez-vous à nouveau avec le navigateur de la machine physique sur les IPs des 2 VMs. Que voyez-vous ?

8 Tests CPU (si il vous reste du temps...)

Nous allons utiliser un benchmark de calcul (donc qui stresse le processeur) basé sur un algorithme de calcul de nombre premier. Usage :

```
sysbench --test=cpu  --cpu-max-prime=10000 --num-threads=1 run | grep "total time:"
```

La commande ci-dessus effectue 1000 calculs indépendants. Si N CPUs sont disponibles, cela devrait donc prendre $\frac{1}{N}$ième du temps pour une seule CPU si vous ajustez le nombre de threads (via le paramètre –num-threads=) à N.

1. Modifier votre VM pour qu'elle ait 2 vCPU.
2. Démarrez un container ubuntu : **docker run -ti --name=ubuntu ubuntu /bin/bash**, installez sysbench et faites tourner le benchmark.
3. Combien de CPU voyez-vous dans le container ?
4. Comparez les performances avec celles en natif de la VM.
5. Stoppez le container et détruisez le.
6. Re-démarrez un container similaire avec la commande :

   ```
   docker run -ti --cpuset-cpus="1" --name=ubuntu ubuntu /bin/bash
   ```

   et refaites le test. Qu'obtenez-vous ?