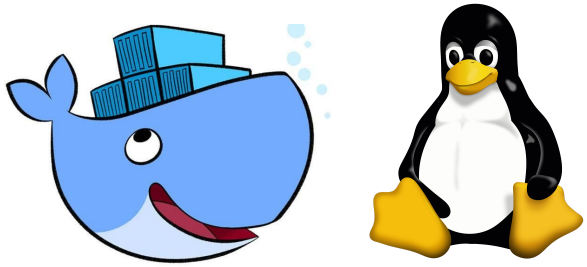

TP
Namespaces et Conteneurs

Date: 09/12/2019

Objectif du TP

Ce TP a pour objectif de vous familiariser avec l'utilisation de conteneurs sous Linux tels que popularisés par Docker ainsi que de vous faire comprendre les mécanismes Linux permettant de les implémenter.

Complétez le google form suivant avec vos réponses:

https://goo.gl/forms/8xfriCYLPLjG6vEg1

Mise en place de l'environnement

Le TP est à réaliser sur le cluster HPC de l'ENSIIE

Commencez par définir quelques variables d'environnement

```
$ export SCRATCHDIR=/scratch/$USER
$ export TPHOME=/scratch/tp/cont
```

Copier le contenu de `$TPHOME/templates.yaml` dans `~/.pcocc/templates.yaml`.

Créez le répertoire `$SCRATCHDIR/tp-cont`.

Copier et adapter le contenu de `$TPHOME/inituser.ci` dans `$SCRATCHDIR/tp-cont/inituser.ci` (remplacez les TODO par vos login et votre clé publique SSH).

Avec la commande `pcocc batch -t 6:00:00 -c 4 docker`, lancez la VM docker et connectez-y vous avec `pcocc ssh vm0`

Pour la suite du TP, travaillez en root dans votre vm (`sudo -i`).

Facultatif: sauvegardez votre VM pour la relancer ensuite:

```
$ pcocc save -d $SCRATCHDIR/tp-cont/mydocker
# Détruisez votre premier job pcocc
$ squeue
$ scancel [jobid]
$ pcocc batch -t 6:00:00 -c 4 mydocker
# Vous pouvez maintenant faire des sauvegardes incrémentales avec
$ pcocc save
```

Prise en main de docker

Placez vous dans le répertoire `/root/cli`. L'objectif de cette partie est d'exécuter le script `prettycow` (disponible dans ce répertoire) dans un conteneur ubuntu contenant les dépendances nécessaires.

- Créez un conteneur Docker à partir de l'image `ubuntu` dans lequel vous lancerez un shell. À partir de ce shell, installez les paquets nécessaires à l'exécution du script `prettycow` (avec le gestionnaire de paquet apt-get). La liste des dépendances nécessaires est indiquée en commentaire dans le script. Copiez le script dans votre conteneur et vérifiez qu'il fonctionne (vous devrez modifier la variable d'environnement $PATH car certains binaires sont installés dans /usr/games). **(Q1.1)**

- Retrouvez l'identifiant de votre conteneur et relancez le (**Q1.2**). Vérifiez que vous pouvez toujours exécuter votre script.

- La commande `docker inspect` permet d'afficher des informations sur tout objet docker à partir de son identifiant. A l'aide de cette commande, trouvez où sont stockés les fichiers de la couche modifiable de votre conteneur sur la machine hôte et retrouvez le script que vous avez copié dans votre conteneur. (**Q1.3**). Faites le lien avec les informations renvoyées par les commandes `mount` et `docker diff`. (**Q1.4**)

- Sauvegardez votre conteneur dans une image et relancez un conteneur à partir de cette image. (**Q1.5**)

- Comparez les namespaces de différents processus avec la commande lsns et déduisez en quels sont les namespaces utilisés par défaut pour isoler des conteneurs Docker. (**Q1.6**)

- Écrivez un Dockerfile permettant d'automatiser la création d'une image contenant ce script. Utilisez la directive `ENTRYPOINT` pour que le script soit exécuté par défaut au lancement du conteneur. (**Q1.7**)

Dockerisation d'une application web

On souhaite exécuter l'application web présente dans `/root/web/`. Il s'agit d'une application python Flask utilisant un serveur MySQL. Nous allons la déployer dans deux conteneurs, le premier hébergeant le serveur web, le second hébergeant la base de données.

- Un README est fourni pour installer l'application. Écrivez un Dockerfile générant une image contenant le serveur web. On ne s'occupe pas pour l'instant de la base de données qui sera instanciée dans un autre conteneur. Par défaut, le serveur web de l'application sera exécuté au lancement d'un conteneur à partir de cette image (**Q2.1**)

- Lancez un conteneur en arrière plan à partir de cette image en vous assurant de rediriger un port de l'hôte vers le port sur lequel le votre serveur écoute dans le conteneur. (**Q2.2**)

- Connectez vous à votre application web à l'aide de `curl` ou d'un navigateur web (`lynx` est installé dans votre VM, vous pouvez aussi utiliser un navigateur graphique). Quelle erreur recevez-vous ? Consulter les logs de votre serveur web pour en diagnostiquer la cause. (**Q2.3**).

- Ce conteneur doit être connecté à une base de données pour fonctionner. Pour cela commencez par créer un réseau Docker qui permettra d'interconnecter plusieurs conteneurs.

Créez ensuite un conteneur serveur mysql en version 5.6. En vous aidant de la documentation de l'image officielle mysql sur DockerHub (https://hub.docker.com/_/mysql/), faites en sorte qu'une base de données soit automatiquement initialisée sur ce serveur (**Q2.4**)

- Relancez un conteneur application web sur ce nouveau réseau et à l'aide des variables d'environnement décrites dans le README, connectez le à votre conteneur mysql. (**Q2.5**)

- Réessayez de vous connecter à votre application. Notez à nouveau l'erreur obtenue et la nouvelle raison dans les logs. Corrigez le problème en lisant le README (**Q2.6**).

- Votre application est maintenant fonctionnelle. Peuplez la base de données en créant quelques étudiants via l'interface web.

- Les données de votre base de données MySQL (/var/lib/mysql) sont elles stockées dans la couche modifiable de votre conteneur ? Utilisez les commandes `docker history`, `docker inspect` et `mount` pour le mettre en évidence. (**Q2.7**)

- Arrêtez et supprimez votre conteneur SQL et recréez un conteneur mysql:5.7 en conservant la BDD. Les formats de données étant incompatible, vous verrez des erreurs dans les logs mysql au lancement. Il faut alors exécuter dans votre conteneur mysql la commande `mysql_upgrade -uroot -p` puis relancer le conteneur (**Q2.8**). Vérifiez que l'application web est toujours fonctionnelle et voit toujours vos étudiants.

- Modifiez légèrement les sources de votre application (par exemple modifiez la chaîne de caractères "Flask SQLAlchemy Example" dans templates/show_all.html) et reconstruisez votre conteneur. Toutes les dépendances du conteneur sont à nouveau téléchargées inutilement. Modifiez votre Dockerfile pour que les dépendances ne soient rechargées qu'en cas de modification du fichier requirements.txt. (**Q2.9**)

- Arrêtez et supprimez vos conteneurs et images (**Q2.10**). Écrivez un fichier docker-compose permettant de déployer base de données et serveur web en une seule commande docker-compose up (**Q2.11**).

Ecriture d'un mini runtime de conteneurs: mocker

- Une image de conteneur peut être générée à partir d'un simple fichier tar contenant l'ensemble du système de fichiers. Pour les distributions basées sur debian, la commande `debootstrap [version] [répertoire]` permet d'installer les paquets minimaux nécessaires à la distribution dans le répertoire cible. Pour fedora, on peut utiliser directement dnf avec les options `--installroot` et `--releasever` et installer le groupe de paquet `@core` (utilisez un chemin absolu pour installroot). Générez deux tarball correspondant aux distributions fedora31 et ubuntu xenial. Gardez aussi les répertoires décompressés pour la suite. (**Q3.1**).

  NB: Préfixer les commandes accédant à internet comme debootstrap des exports de variable suivants pour pour utiliser le proxy de l'école (dans la VM, dnf et docker sont préconfigurés pour utiliser ces proxys) `http_proxy=http://hpc01.c-hpc.pedago.ensiie.fr:3128` `https_proxy=http://hpc01.c-hpc.pedago.ensiie.fr:3128`

- En utilisant le système de fichiers Overlay, montez un rootfs qui sera utilisable par un conteneur en combinant, pour le `lowerdir` un des répertoire contenant le rootfs ubuntu ou fedora que vous venez de créer, et des répertoires vierges pour les `workdir` et `upperdir`. (**Q3.2**)

- Créez un bridge nommé `mocker0`, choisissez une plage réseau et assignez lui une IP dans cette plage. Ajoutez une règle iptables MASQUERADE pour router à l'aide de NAT les paquets sortant de cette interface (`iptables -A POSTROUTING -t nat -j  MASQUERADE -s [plage ip] ! -o [nom du bridge]; iptables -I FORWARD 1 -o [nom du bridge] -j ACCEPT; iptables -I FORWARD 1 -i [nom du bridge] -j ACCEPT`). (**Q3.3**)

- Créez un namespace réseau persistent avec `ip netns add [nsname]`. Créez une paire veth que vous connecterez d'un côté à votre bridge et dont vous assignerez l'autre côté à votre namespace réseau. Avec la commande `ip netns exec [nsname] [cmd]`, lancez un shell dans votre namespace réseau et configurez l'autre côté de votre interface veth. Avec `ip route`, positionnez l'IP de `mocker0` comme route par défaut. Vérifiez que vous pouvez pinger l'IP assignée à `mocker0` ainsi que `hpc01.c-hpc.pedago.ensiie.fr`. (**Q3.4**)

- Au sein de votre namespace réseau, avec la commande `unshare`, exécutez un shell dans de nouveaux namespaces de PID, mount, IPC et UTS. Pour la création d'un namespace de PID, il faut ajouter l'option `--fork` à unshare. Que constatez vous en faisant un `ps faux` ? La commande ps tire ses informations du contenu de `/proc` qui est un montage du système de fichiers synthétique `proc`. Or comme vous pouvez le constater en tapant la commande mount, les montages d'un nouveau namespace `mount` sont hérités du namespace parent. A l'aide la commande , montez le système de fichiers synthétique `proc` correspondant à notre nouvel espace de PID. Comme nous sommes aussi dans un nouveau namespace de mount, cela n'affecte pas ce que voient les autres processus dans `/proc`.

- A l'aide de `pivot_root`, déplacer la racine du filesystem dans le rootfs que vous aviez monté à l'aide du système de fichier Overlay (consultez le man). (**Q3.5**) Félicitations, vous avez créé manuellement un conteneur !

- Ecrire un script mocker automatisant et combinant toutes les étapes de création d'un conteneur vues ci-dessus. (**Q3.6**) On suppose toutefois que le bridge et la règle iptables de routage NAT ont été créés manuellement à l'avance. La commande mocker doit permettre de réaliser les commandes suivantes:
    - mocker import [tarball] [imagename] : Importe un tarball sous le nom d'image *imagename*.
    - mocker run [imagename] [cmd....]: Crée un conteneur à partir d'une image préalablement importée et exécute la commande. (lui assigner un identifiant aléatoire).
    - mocker ps: Liste tous les conteneurs créés.
    - mocker rm [containerid]: Supprime un conteneur.
    - mocker commit [containerid] [imagename]: Crée une nouvelle image à partir d'un conteneur.
- **Bonus**: vos conteneurs ne sont pas complètement isolés car l'utilisateur root au sein du conteneur est le même que sur le système hôte. Sécurisez vos conteneurs en utilisant un user namespace.