Département Informatique
RA-4.08 Cybersécurité – TP Docker
Responsable : X.Roirand + M. Le Lain
Durée : 90 mn (sur machine)

Manipulation Docker

Le but de ce TP va être de manipuler des instances docker et/ou des containers et d'apprivoiser les commandes de base.

Attention : certaines commandes comme le démarrage d'un container peuvent prendre parfois jusqu'a 30 secondes, voir un peu plus, soyez patient :)

1) Connection sur la machine distante ou on doit travailler

Pour se connecter sur la machine distante, il faut utiliser une connexion ssh. C'est une connexion sécurisée qui permet de ne pas faire passer le mot de passe en clair sur le réseau.

Vous pouvez utiliser le programme « putty » sur windows ou « ssh » sur linux.

La connexion se fait sur une machine dont l'adresse IP est 195.83.161.135. Pour que chaque étudiant ait sa propre machine, les ports ssh de connexion diffèrent. Lors de la mise en place du TP, l'enseignant vous a donné un numéro. Ce numéro est important car il va être utilisé pour le port ssh de connexion et le mot de passe.

Pour résumer:

IP: 149.202.85.73
port: 122<votre numéro sur 2 digits>
login: student
mot de passe est: student

Ex:

Si votre numéro est 86 alors le port sera 12286

Une fois sur votre machine distante, vous pouvez l'administrer totalement. Vous devez absolument changer le mot de passe par défaut le plus rapidement possible.

Pour pouvoir noter le TP, il faut que vous mettiez dans un fichier nommé /root/info.txt votre nom et prénom. Attention, pas de fichier = zéro !

Tout ce qui est noté sur fond bleu … (xxx réponse à fournir)…correspond à une question à laquelle vous devez répondre dans le document de rendu.

A la fin de la séance, vous devez rendre ce document dans la zone de rendu « Rendu séance », même si vous n'avez pas fini le TP.

2) **Manipulation de container (suite):**

Dans le context du TP précédent, dites comment on peut constater qu'on est bien dans un container (guest OS) et de type Alpine, ou tout autre container, et pas dans la machine initiale qui vous est assignée.

Mettez cette information dans le document de rendu (1ère réponse à fournir).

```
student@test:~$ cat /etc/os-release
NAME="Ubuntu"
VERSION="18.04 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04 LTS"
VERSION_ID="18.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-
policies/privacy-policy"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
student@test:~$ ls -la /.dockerenv
ls: cannot access '/.dockerenv': No such file or directory
student@test:~$
student@test:~$ hostname
test
```

A priori on est bien sur une machine Ubuntu 18.04 et non dans un docker Alpine d'après les informations ci-dessus.

Lancez un container Alpine en mode interactif, puis quittez le (Attention, pas forcément évident en fonction de la commande utilisé pour lancer le container).

Dites comment vous avez fait pour quitter dans le document de rendu (2ème réponse à fournir).

```
student@test:~$ docker run -it alpine /bin/sh
/ #
/ # ls
bin    etc    lib    mnt    proc   run    srv    tmp    var
dev    home   media  opt    root   sbin   sys    usr
/ #
/ # exit
student@test:~$
```

Dans la question suivate, on va lancer le même container mais cette fois ci en arrière-plan et non-interactif. Pour vérifier que le container tourne bien en arrière-plan, nous allons faire (sur une seule ligne de commande), un shell qui tourne en boucle et affiche une chaine de caractère et aussi qui fait une pause de 1 seconde entre chaque boucle sinon votre processeur va être occupé à 100% !

Avant de traiter cette question, posons-nous la question de pourquoi lancer l'instance docker en arrière plan avec un shell qui boucle sur lui même, au lieu juste de lancer un le container avec le shell a exécuter (3ème réponse à fournir) ?

Un container Docker s'arrête dès que son processus principal se termine.
Lancer un shell avec une boucle infinie permet :

- De garder le container vivant
- De pouvoir ensuite consulter ses logs

Sans boucle, le container se terminerait immédiatement.

Maintenant pour traiter la question, je vous propose de commencer par faire la ligne de commande en Shell et la tester sur la machine qui vous est assignée. Une fois que cela fonctionne, on peut ensuite tester en lançant la commande Shell via le container docker.

Mettez cette commande et la sortie écran produite lorsque vous avez rentré cette commande dans le document de rendu (4ème réponse à fournir).

```
student@test:~$ docker run -d alpine sh -c 'while true; do echo "Container actif"; sleep 1; done'
f52fa6b2b10ae31b5c5a8840b61b4a55668d0192aba90e0baf94a7d6047adffc

student@test:~$
```

Constatez qu'aucun téléchargement ne se fait car l'image a déjà été téléchargée lors de l'exercice précédent et a été conservé dans un cache.

Trouvez maintenant les commandes docker qui permettent de :

- Afficher la liste des containers actifs (5ème réponse à fournir)

```
student@test:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS     NAMES
f52fa6b2b10a   alpine    "sh -c 'while true; …"   9 seconds ago   Up 7 seconds             epic_brattain
```

- Afficher l'ensemble des containers (y compris ceux qui sont stoppés ou terminés) (6ème réponse à fournir).

```
student@test:~$ docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED         STATUS                     PORTS     NAMES
f52fa6b2b10a   alpine               "sh -c 'while true; …"   2 minutes ago   Up 2 minutes                         epic_brattain
a36589404535   alpine               "/bin/sh"                7 minutes ago   Exited (0) 7 minutes ago             awesome_dirac
79cbf5f46792   eclipse-temurin:17   "/__cacert_entrypoin…"   6 days ago      Exited (0) 6 days ago                modest_mendeleev
f3c0030accce   nickblah/lua         "/bin/sh"                6 days ago      Exited (0) 6 days ago                distracted_brahmagupta
47d7318e5b3b   alpine               "/bin/sh"                6 days ago      Exited (0) 6 days ago                adoring_sinoussi
cb11c0e6ce8b   alpine               "/bin/sh"                6 days ago      Exited (0) 6 days ago                elated_bhabha
913a015613fe   php                  "docker-php-entrypoi…"   6 days ago      Exited (0) 6 days ago                distracted_kare
213541065b62   node                 "docker-entrypoint.s…"   7 days ago      Exited (0) 7 days ago                interesting_carver
e1aedc563b02   node                 "docker-entrypoint.s…"   7 days ago      Exited (0) 7 days ago                fervent_mayer
c9319588a4f7   python:3             "/bin/bash"              7 days ago      Exited (0) 7 days ago                nervous_varahamihira
```

Mettre toutes ces commandes et les sorties écrans associés dans le document de rendu.

Cherchez comment afficher la sortie écran (output) du container précédemment lancé, et mettez la commande et l'output dans le document de rendu (7ème réponse à fournir).

```
student@test:~$ docker logs epic_brattain
Container actif
Container actif
Container actif
Container actif
Container actif
Container actif
Container actif
Container actif
Container actif
…
Container actif
Container actif
Container actif
Container actif
```

Trouvez comment stopper un container et mettez la commande dans le document de rendu (8 ème réponse à fournir).

```
student@test:~$ docker stop epic_brattain
epic_brattain
student@test:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS
NAMES
student@test:~$
```

Lorsqu'on fait des opérations sur un container docker, il est possible de nommer le container docker de plusieurs manières (son nom, etc…).

Quels sont les identifiants possibles qu'on peut utiliser pour préciser le container lorsqu'on veut l'arrêter ?

Mettez ces identifiants dans le document de rendu (9ème question) (Attention, on ne demande pas ici l'identifiant précis de votre container, mais d'une manière générale avec quels éléments on peut identifier un container pour des opération de start, stop , etc…)

On peut identifier un container par :

- ID long (ex: a7c3d1f9b82e5a0f...)
- ID court (ex: a7c3d1f9b82e)
- Nom du container (ex: my_container)
- Nom généré automatiquement par Docker (ex: epic_brattain)

Ces identifiants peuvent être utilisés pour start, stop, logs, attach, etc.

Maintenant on vous demande de redémarrer le container que vous avez arrêté précédemment et de vous y attacher (Attention: parfois il faut faire un retour chariot pour que vous récupériez la main dans votre container).

Quelle commande avez-vous fait pour relancer le container arrêté précédemment ? (10ème question)

```
student@test:~$ docker start epic_brattain
epic_brattain
student@test:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
f52fa6b2b10a   alpine    "sh -c 'while true; …"   11 minutes ago   Up 43 seconds             epic_brattain
student@test:~$
```

Quelle commande avez-vous fait pour vous attacher au container ? (11ème question)

```
student@test:~$ docker attach epic_brattain
Container actif
Container actif
Container actif
Container actif
Container actif
Container actif
Container actif
Container actif
^Cstudent@test:~$
```

Vérifier en utilisant la commande qui permet de lister tous les containers, que votre container a bien démarré.

Mettre la commande de verification dans le document de rendu (12ème réponse à fournir).

```
student@test:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                      PORTS     NAMES
f52fa6b2b10a   alpine    "sh -c 'while true; …"   15 minutes ago   Exited (130) 2 minutes ago            epic_brattain
```

On va maintenant chercher une image de la distribution Busybox.

Trouvez la/es commande(s) qui permettent de la télécharger seulement (et pas de l'exécuter).

Mettez cette/ces commande(s) et la sortie écran produite dans le document de rendu (13ème réponse à fournir).

```
student@test:~$ docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox
61dfb50712f5: Pull complete
Digest:
sha256:b3255e7dfbcd10cb367af0d409747d511aeb66dfac98cf30e97e87e4207dd76f
Status: Downloaded newer image for busybox:latest
docker.io/library/busybox:latest
student@test:~$
```

Busybox est utilisé dans des systèmes ou on a besoin d'un shell minimal, essayez maintenant de lancer le container busybox avec le shell bash. Est-ce que ça fonctionne ?

Expliquez le résultat dans le document de rendu (14ème réponse à fournir).

```
student@test:~$ docker start angry_colden
Error response from daemon: failed to create task for container: failed to create shim task: OCI runtime create failed: runc create failed:
unable to start container process: exec: "bash": executable file not found in $PATH: unknown
Error: failed to start containers: angry_colden
student@test:~$
student@test:~$ docker ps -a
CONTAINER ID   IMAGE              COMMAND                  CREATED             STATUS                         PORTS     NAMES
8ada0c8de798   busybox            "bash"                   3 minutes ago       Created                                  angry_colden
```

Lors du start, Docker tente d'exécuter la commande par défaut : « bash »
Mais, le lancement de BusyBox avec bash ne fonctionne pas car sur BusyBox le shell bash n'y est pas installé, seul le shell sh est disponible.

Maintenant que vous en savez un peu plus sur Busybox, lancez un shell en interactif sur le container Busybox.

Mettez cette commande et la sortie écran produite lorsque vous avez rentré cette commande dans le document de rendu (15ème réponse à fournir).

```
student@test:~$ docker run -it busybox sh
/ #
/ # exit
student@test:~$
```

Nous allons maintenant essaye de modifier une image existante, pour cela on va faire plusieurs étapes:

- Récupérer l'image qu'on veut modifier
- Lancer l'image de manière interactive pour etre dans un container
- Modifier le contenu de ce container
- Sauvegarder le container en image avec un nom que vous choisissez

Essayez de faire ca sur une image busybox, en rajoutant dans le container, le fichier /home/essai.txt

Expliquez dans le document de rendu comment vous avez fait ? (16ème réponse à fournir).

```
student@test:~$ docker run -it busybox sh
/ # mkdir -p /home
/ # echo "Test modification image" > /home/essai.txt
/ # exit
student@test:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS                      PORTS     NAMES
8d6c1666ca33   busybox   "sh"      28 seconds ago   Exited (0) 23 seconds ago             eager_jang
student@test:~$ docker commit 8d6c1666ca33 busybox_modifie
sha256:f67ca4474544a756c4cb47bdf0579a13ba79c748ebe369f4b0c41587be2522a7
student@test:~$
```

Bien sûr, il faut tester si l'image que vous avez modifiée a vraiment été modifié, pour cela lancer un container à partir de l'image modifiée, et regardez si le fichier /home/essai.txt est bien présent.

Expliquez dans le document de rendu les différentes étapes de vérification. (17ème réponse à fournir).

```
student@test:~$ docker run -it busybox_modifie sh
/ # ls /home
essai.txt
/ # cat /home/essai.txt
Test modification image
/ # exit
student@test:~$
```

Vous allez maintenant effacer l'image modifiée que vous avez créé. Indiquez la commande utilisée pour trouvez et effacer cette image. (18ème réponse à fournir).

```
student@test:~$ docker images
REPOSITORY        TAG       IMAGE ID       CREATED          SIZE
busybox_modifie   latest    f67ca4474544   2 minutes ago    4.43MB
alpine            latest    a40c03cbb81c   9 days ago       8.44MB
node              latest    a6f50e7139f3   9 days ago       1.13GB
php               latest    d79cfd3ccda5   2 weeks ago      571MB
eclipse-temurin   17        53c2311899d3   3 weeks ago      420MB
python            3         76a545d97be5   3 weeks ago      1.12GB
nickblah/lua      latest    04e7b76cafe7   3 weeks ago      122MB
busybox           latest    af3f0f48a24e   16 months ago    4.43MB
student@test:~$ docker rmi busybox_modifie
Untagged: busybox_modifie:latest
Deleted:
sha256:f67ca4474544a756c4cb47bdf0579a13ba79c748ebe369f4b0c41587be2522a7
Deleted:
sha256:fdead2fce242ca63a3194d9ed936657792a6cd869a079a08a6971bdf8b631114
student@test:~$
```

Relancez un container busybox et vérifier que c'est bien l'image d'origine qui s'est lancé et pas celle modifiée (qui ne doit plus exister).

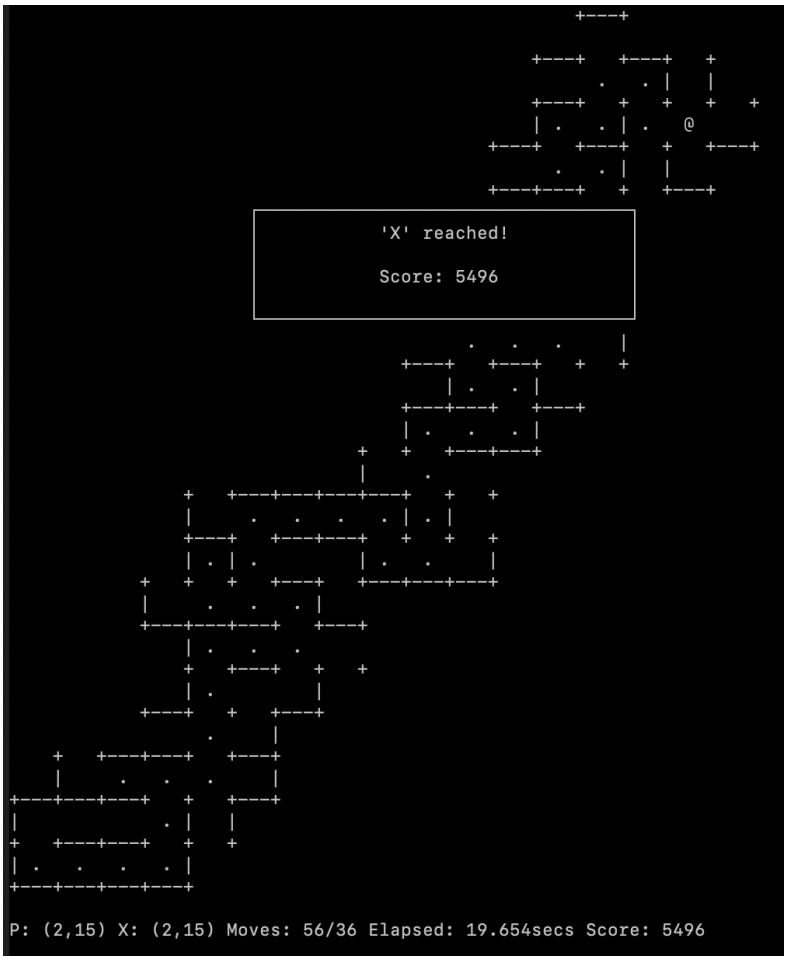
```
student@test:~$ docker run -it busybox sh
/ # ls /home
/ # cat /home/essai.txt
cat: can't open '/home/essai.txt': No such file or directory
/ # exit
student@test:~$
```

Pour rappel, un container n'est ni plus ni mois qu'un système de fichier, trouvez quelle commande docker utiliser pour lister le contenu des fichiers présent dans une image docker. ATTENTION, il ne faut pas lancer d'instance de l'image docker. (19ème réponse à fournir).

```
student@test:~$ docker save busybox -o busybox.tar
student@test:~$ tar -tf busybox.tar
8ecce086fb746dcb49a7fdd410c7a9d871f28bdcd80fab0d7f03fbf9dc17d9fb/
8ecce086fb746dcb49a7fdd410c7a9d871f28bdcd80fab0d7f03fbf9dc17d9fb/VERSION
8ecce086fb746dcb49a7fdd410c7a9d871f28bdcd80fab0d7f03fbf9dc17d9fb/json
8ecce086fb746dcb49a7fdd410c7a9d871f28bdcd80fab0d7f03fbf9dc17d9fb/layer.tar
af3f0f48a24edb84e94aff6f44f5d089203453719d3b2328486d311e61db9b09.json
manifest.json
repositories
student@test:~$ tar -xf busybox.tar
student@test:~$ find . -name layer.tar -exec tar -xf {} \;
student@test:~$ ls
8ecce086fb746dcb49a7fdd410c7a9d871f28bdcd80fab0d7f03fbf9dc17d9fb       busybox.tar  home   manifest.json  tmp
af3f0f48a24edb84e94aff6f44f5d089203453719d3b2328486d311e61db9b09.json  dev          lib    repositories   usr
bin                                                                    etc          lib64  root           var
student@test:~$
```

Faire fonctionner le jeu https://hub.docker.com/r/kazhar/mazingame/ . Expliquez sur le document de rendu ce que vous avez fait pour que cela fonctionne. (20ème réponse à fournir).

```
student@test:~$ docker pull kazhar/mazingame
Using default tag: latest
latest: Pulling from kazhar/mazingame
4176fe04cefe: Pull complete
851356ecf618: Pull complete
6115379c7b49: Pull complete
aaf7d781d601: Pull complete
40cf661a3cc4: Pull complete
975fe2fd635f: Pull complete
bf4db784e7fd: Pull complete
0491f7e9426b: Pull complete
75eeb9a45c2d: Pull complete
31c2bcd9cbba: Pull complete
de7e3953bc13: Pull complete
123400c74fb1: Pull complete
944f3b306923: Pull complete
847b7e6594ce: Pull complete
Digest:
sha256:f3186ad7998f52bb5b133079c06fdb11643c1a1efd9596d42e287c1b378d76
59
Status: Downloaded newer image for kazhar/mazingame:latest
docker.io/kazhar/mazingame:latest
student@test:~$
student@test:~$ docker run -it kazhar/mazingame
'X' reached:
  Game ID   : 1
  Level     : 791010016
  Algorithm: Aldous Broder
  Moves     : 56/36
  Elapsed   : 19.654secs
  Score     : 5496
student@test:~$
```


'X' reached!
Score: 5496
P: (2,15) X: (2,15) Moves: 56/36 Elapsed: 19.654secs Score: 5496