Les commandes de base de gestion des containers

Exercice 1 : Hello from Alpine

Le but de ce premier exercice est de lancer des containers basés sur l'image alpine

1. Lancez un container basé sur alpine en lui fournissant la command echo hello
2. Quelles sont les étapes effectuées par le docker daemon ?
3. Lancez un container basé sur alpine sans lui spécifier de commande. Qu'observez-vous ?

Correction de l'exercice 1

1. La commande à lancer est

```
docker container run alpine echo hello
```

2. Le résultat de la commande précédente montre les différentes étapes qui sont effectuées:

```
Unable to find image 'alpine:3.8' locally
latest: Pulling from library/alpine
88286f41530e: Pull complete
Digest: sha256:1072e499f3f655a032e88542330cf75b02e7bdf673278f701d7ba61629ee3ebe
Status: Downloaded newer image for alpine:latest
hello
```

L'image alpine n'étant pas disponible localement, elle est téléchargée depuis le Docker Hub. Un container est alors lancé à partir de cette image avec la commande spécifiée (echo "hello"). Le résultat de cette commande est affiché sur la sortie standard. Une fois la commande terminée, le container est supprimé.

3. La commande à lancer est "docker container run alpine"

L'image alpine étant déjà présente localement (téléchargée lors de l'étape précédente), elle est ré-utilisée. La commande exécutée par défaut lors du lancement d'un container basé sur

alpine est "/bin/sh". Cette commande ne produit pas de résultat sur la sortie standard. Le container est supprimé une fois la commande lancée.

Exercice 2 : shell intéractif

Le but de cet exercice est lancer des containers en mode intéractif

1. Lancez un container basé sur alpine en mode interactif sans lui spécifier de commande
2. Que s'est-il passé ?
3. Quelle est la commande par défaut d'un container basé sur alpine ?
4. Naviguez dans le système de fichiers
5. Utilisez le gestionnaire de package d'alpine (apk) pour ajouter un package

```
$ apk update
$ apk add curl
```

Correction de l'exercice 2

1. La commande permettant de lancer un container en mode "interactif" est la suivante:

```
docker container run –ti alpine
```

2. Nous obtenons un shell sh dans le container.
3. Par défaut, la commande par défaut utilisée dans l'image alpine est /bin/sh

Note: nous y reviendrons plus tard mais il est intéressant de voir que cette information est présente dans le fichier Dockerfile qui est utilisé pour créer l'image (https://github.com/gliderlabs/docker-alpine/blob/2127169e2d9dcbb7ae8c7eca599affd2d61b49a7/versions/library-3.6/x86_64/Dockerfile)

4. Nous pouvons naviguer dans le système de fichiers de la même façon que nous le faisons

dans une autre distribution Linux plus "traditionnelle" en utilisant les commandes cd, ls, pwd, cat, less, …

5. Le gestionnaire de package d'une distribution alpine est apk

Pour mettre à jour la liste des packages, nous pouvons utiliser la commande apk update .

Pour installer un package, comme curl , nous utilisons la commande suivante apk add curl

---

Exercice 3 : foreground / background

Le but de cet exercice est de créer des containers en foreground et en background

1. Lancez un container basé sur alpine en lui spécifiant la commande `ping 8.8.8.8`
2. Arrêter le container avec CTRL-C

Le container est t-il toujours en cours d'exécution ?

Note: vous pouvez utiliser la command docker ps que nous détaillerons dans l'une des prochaines lectures), et qui permet de lister les containers qui tournent sur la machine.

3. Lancez un container en mode interactif en lui spécifiant la command `ping 8.8.8.8`
4. Arrêter le container avec CTRL-P CTRL-Q

Le container est t-il toujours en cours d'exécution ?

5. Lancez un container en background, toujours en lui spécifiant la command `ping 8.8.8.8`

Le container est t-il toujours en cours d'exécution ?

Correction de l'exercice 3

1. Le container peut être lancé avec la commande suivante

```
docker container run alpine ping 8.8.8.8
```

2. La commande docker ps ne liste pas le container car le processus a été arrêté par la

commande CTRL-C

3. La commande suivante permet de lancer le container en mode "interactif"

```
docker container run -ti alpine ping 8.8.8.8
```

4. La commande CTRL-P CTRL-Q permet de se détacher du speudo terminal (alloué avec les options -t -i, ou -ti).

Le container continue à tourner. Il est listé avec la commande `docker ps`

5. La commande suivante permet de lancer le container en background

```
docker container run -d alpine ping 8.8.8.8
```

La commande docker ps permet de voir que le container tourne, il n'est simplement pas attaché au terminal depuis lequel il a été lancé.

Exercice 4 : publication de port

Le but de cet exercice est de créer un container en exposant un port sur la machine hôte

1. Lancez un container basé sur nginx et publiez le port 80 du container sur le port 8080 de l'hôte

2. Vérifiez depuis votre navigateur que la page par défaut de nginx est servie sur http://localhost:8080

3. Lancez un second container en publiant le même port

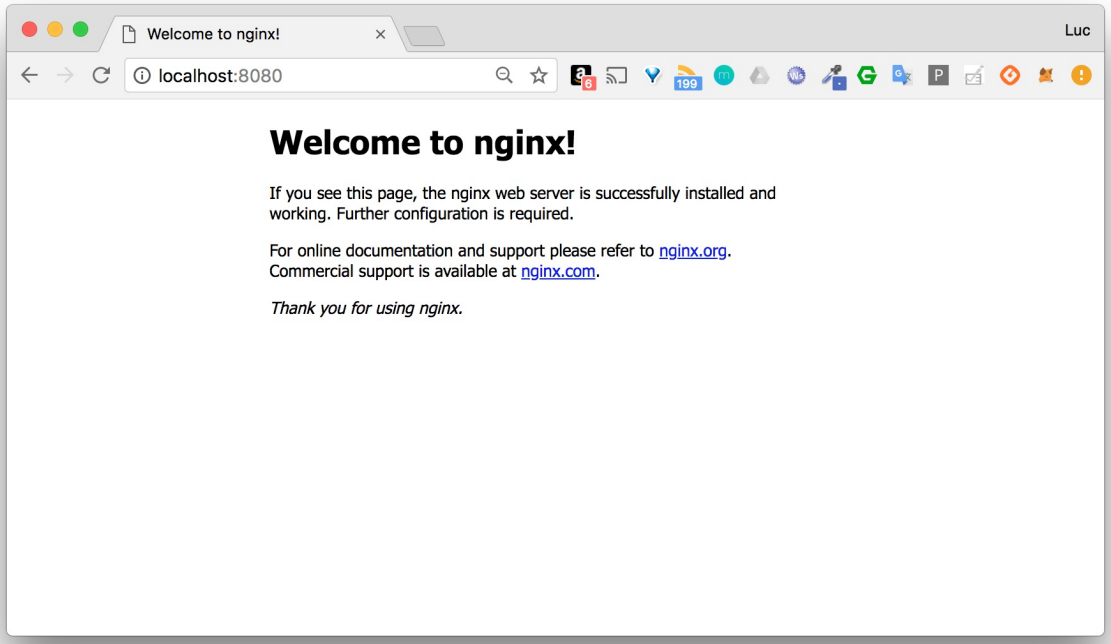
Qu'observez-vous ?

Correction de l'exercice 4

1. La commande suivante permet de lancer le container basé sur nginx et d'exposer le port 80 de ce container sur le port 8080 de la machine hôte

```
docker container run -d -p 8080:80 nginx
```

2. La page index.html servie par défaut par nginx est accessible sur le port 8080



3. Si nous lançons un autre container en utilisant le même port de la machine hôte, nous obtenons une erreur car ce port ne peut pas être utilisé pour les 2 containers.

```
$ docker container run -d -p 8080:80 nginx
c9dc92128bd8871d1c75678fd41dc09c5afcf02857c7c64bd89f560cb2b6aec7
docker: Error response from daemon: driver failed programming external
connectivity on endpoint objective_leakey
(25ea6fe30398e51b5ad3dbd55a56cded518370c81a41d4013d58ca399647718a): Bind for
0.0.0.0:8080 failed: port is already allocated.
```

Exercice 5 : liste des containers

Le but de cet exercice est de montrer les différentes options pour lister les containers du système

1. Listez les containers en cours d'exécution

Est ce que tous les containers que vous avez créés sont listés ?

2. Utilisez l'option -a pour voir également les containers qui ont été stoppés

3. Utilisez l'option -q pour ne lister que les IDs des containers (en cours d'exécution ou stoppés)

Correction de l'exercice 5

1. Pour lister les containers en cours d'execution sur le système, les 2 commandes suivantes sont équivalentes

- `docker container ls`
- `docker ps`

Note: la seconde commande est historique et date de l'époque ou la plateforme Docker était principalement utilisée pour la gestion des containers et des images. Par la suite, d'autres primitives que nous verrons dans les prochains chapitres ont été ajoutées (volume, network, node, …), ce qui a conduit à une refactorisation de l'api disponible en ligne de commande.

2. Seuls les containers en cours d'execution sont listés. Les containers qui ont été créés puis arrêtés ne sont pas visible dans la sortie de cette commande.

3. Les commandes suivantes permettent de lister les containers en cours d'execution ainsi que ceux qui sont stoppés.

```
docker container ls –a  et docker ps –a
```

4. Les commandes suivantes permettent de lister les identifiants de l'ensemble des containers tournant sur la machine hôte, ceux en cours d'execution et également ceux qui ont été stoppés.

- `docker container ls –aq`
- `docker ps –aq`

Note: on verra que ces commandes sont très utiles, notamment lorsque l'on souhaitera faire une action sur un ensemble de containers.

Exercice 6 : inspection d'un container

Le but de cet exercice est l'inspection d'un container

1. Lancez, en background, un nouveau container basé sur nginx:1.14 en publiant le port 80 du container sur le port 3000 de la machine host.

Notez l'identifiant du container retourné par la commande précédente.

2. Inspectez le container en utilisant son identifiant

3. En utilisant le format Go template, récupérez le nom et l'IP du container

4. Manipuler les Go template pour récupérer d'autres information

Correction de l'exercice 6

1. La commande permettant de lancer le container en question est la suivante

```
$ docker container run -d -p 3000:80 nginx:1.14
```

Vous devriez obtenir un résultat proche de celui ci-dessous:

```
Unable to find image 'nginx:1.14' locally
1.14: Pulling from library/nginx
177e7ef0df69: Pull complete
132d4353ecd5: Pull complete
9482632c8a8f: Pull complete
Digest: sha256:98f78a1dde1ba9c9fbb10671b14a74fcff97f0cc85c182e217618397fcaf63fa
Status: Downloaded newer image for nginx:1.14
6eea1906d21fa26c6bd8695c317e29f99e651657063df1964a18906091bd1ed5
```

2. L'inspection d'un container se fait à l'aide de la commande docker inspect CONTAINER_ID

Note: il est possible de n'utiliser que les permiers caractères de l'identifiant, ou bien le nom du container si celui-ci à été précisé avec l'option --name lors de la création.

```
$ docker inspect 6ee
```

Le résultat de la commande ci-dessus a été volontairement tronqué pour améliorer la lisibilité.


```
[
    {
        "Id": "6eea1906d21fa26c6bd8695...1964a18906091bd1ed5",
        "Created": "2019-01-07T12:43:38.035470972Z",
        "Path": "nginx",
        "Args": [
            "-g",
            "daemon off;"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 6880,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2019-01-07T12:43:39.051210644Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:3f55d5bb33f3ae6e7f232c82...69324c95ac1cb9d7ae",
        "ResolvConfPath":
"/var/lib/docker/containers/6eea1906d21fa26c6bd8695...906091bd1ed5/resolv.conf",
        "HostnamePath":
"/var/lib/docker/containers/6eea1906d21fa26c6bd8695...906091bd1ed5/hostname",
        "HostsPath":
"/var/lib/docker/containers/6eea1906d21fa26c6bd8695...906091bd1ed5/hosts",
        "LogPath": "/var/lib/docker/containers/6eea...1bd1ed5/6eea...1bd1ed5-
json.log",
        "Name": "/loving_proskuriakova",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "docker-default",
        "ExecIDs": null,
        "HostConfig": {
           ...
        },
        "GraphDriver": {
           ...
        },
        "Mounts": [],
        "Config": {
           ...
        },
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID": "8731051602d765fe4c3da4...3ec3439c472d03098",
```

```
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {
                "80/tcp": [
                    {
                        "HostIp": "0.0.0.0",
                        "HostPort": "3000"
                    }
                ]
            },
            "SandboxKey": "/var/run/docker/netns/8731051602d7",
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID": "85d94006c9daee54a1702...d7674ad22aad4529dc92905",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.3",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:03",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "445eea139d71bcffcb...1788775b00a7791ab",
                    "EndpointID": "85d94006c9daee54a...aad4529dc92905",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.3",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:03",
                    "DriverOpts": null
                }
            }
        }
    }
]
```


3. Les format de templating "Go template" permet de récupérer seulement les informations dont on a besoin dans cette imposante structure json.

La commande utilisée pour récupérer le nom du container:

```
$ docker inspect --format "{{ .Name }}" efc940
/elated_mclean
```

La commande pour récupérer l'adresse IP du container:

```
$ docker inspect -f '{{ .NetworkSettings.IPAddress }}' efc940
172.17.0.5
```

4. Le templating Go est très riche, vous trouverez quelques exemples supplémentaires à l'adresse suivante
   Go template

Exercice 7 : exec dans un container

Le but de cet exercice est de montrer comment lancer un processus dans un container existant

1. Lancez un container en background, basé sur l'image alpine. Spécifiez la commande `ping 8.8.8.8` et le nom ping avec l'option --name

2. Observez les logs du container en utilisant l'ID retourné par la commande précédente ou bien le nom du container

Quittez la commande de logs avec CTRL-C

3. Lancez un shell sh, en mode interactif, dans le container précédent

4. Listez les processus du container

Qu'observez vous par rapport aux identifiants des processus ?

Correction de l'exercice 7

1. La commande suivante permet de lancer le container en question

```
$ docker container run -d --name ping alpine ping 8.8.8.8
172c401915f56e3fb10391259fac77bc2d3c194a1b27fa5072335e04656e57bb
```

2. La commande suivante permet de suivre les logs en continue (option -f) à partir de

l'identifiant

```
$ docker container logs -f 172
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=37 time=20.713 ms
64 bytes from 8.8.8.8: seq=1 ttl=37 time=19.747 ms
64 bytes from 8.8.8.8: seq=2 ttl=37 time=19.730 ms
64 bytes from 8.8.8.8: seq=3 ttl=37 time=20.345 ms
64 bytes from 8.8.8.8: seq=4 ttl=37 time=24.476 ms
```

Le nom du container peut également être utilisé

```
$ docker container logs -f ping
...
64 bytes from 8.8.8.8: seq=162 ttl=37 time=20.364 ms
64 bytes from 8.8.8.8: seq=163 ttl=37 time=20.822 ms
64 bytes from 8.8.8.8: seq=164 ttl=37 time=22.478 ms
64 bytes from 8.8.8.8: seq=165 ttl=37 time=19.772 ms
64 bytes from 8.8.8.8: seq=166 ttl=37 time=19.630 ms
```

La commande CTRL-C permet de quitter la sortie de logs, mais elle ne stoppe pas le container car elle n'est pas envoyé au processus tournant dans celui-ci.

3. La commande suivante permet de lancer un shell sh dans le container

```
$ docker exec -ti ping sh
/ #
```

4. La commande suivante permet de lister les processus qui tournent dans le container

```
/ # ps aux
PID   USER     TIME   COMMAND
    1 root       0:00 ping 8.8.8.8
    7 root       0:00 sh
   13 root       0:00 ps aux
```

On peut voir que la commande avec laquelle le container a été lancé (ping 8.8.8.8) à le PID 1 (identifiant du processus). La commande sh que nous avons ensuite lancée dans le container à le PID 7 dans l'arbre de processus. Nous voyons également la commande ps aux qui a obtenue le PID 13, cette commande n'est plus active et si on la relance une nouvelle fois, on

obtiendra un nouveau PID:

```
/ # ps aux
PID   USER     TIME   COMMAND
    1 root       0:00 ping 8.8.8.8
    7 root       0:00 sh
   14 root       0:00 ps aux
```

On peut sortir de notre shell avec un simple `exit`, le container continuera à tourner tant que le process de PID 1 est actif.

Exercice 8 : cleanup

Le but de cet exercice est de stopper et de supprimer les containers existants

1. Listez tous les containers (actifs et inactifs)
2. Stoppez tous les containers encore actifs en fournissant la liste des IDs à la commande stop
3. Vérifiez qu'il n'y a plus de containers actifs
4. Listez les containers arrêtés
5. Supprimez tous les containers
6. Vérifiez qu'il n'y a plus de containers

Correction de l'exercice 8

1. Afin de lister les containers qui tournent ainsi que ceux qui ont été arrêtés, il faut utiliser l'option -a , les commandes suivantes sont équivalentes:

```
docker ps -a
```

```
docker container ls -a
```

2. Pour stopper, en une seule ligne de commande, l'ensemble des containers qui tournent, on peut donner la liste des identifiants à la commande stop . On utilise pour cela l'option -q lorsque l'on liste les containers.

```
docker container stop $(docker container ls –q)
```

3. La commande suivante ne devrait plus renvoyer de containers

```
docker container ls
```

Note: cette commande est équivalente à `docker ps`

4. Si on ajoute l'option -a , on obtient les containers qui sont arrêtés.

```
docker container ls –a
```

Note: cette commande est équivalente à `docker ps –a`

5. Pour supprimer les containers qui sont arrêtés, on procède de la même façon que dans la question 2, on donne la liste des identifiants à la commande rm.

```
docker container rm $(docker container ls –aq)
```

6. La commande suivante ne devrait plus lister aucun container

```
docker container ls –a
```

En résumé

Nous avons commencé à jouer avec les containers et vu les commandes les plus utilisées pour la gestion du cycle de vie des containers (run, exec, ls, rm, inspect). Nous les utiliserons souvent dans la suite du cours.