Activité n°6: Les réseaux

Points important

- Au démarrage du daemon docker:
  - création du bridge ”docker0”
  - une adresse IP privé ainsi qu’une adresse MAC sont assignées au bridge
- Réseau par défaut: 172.17.0.0/16
- Création d’un Bridge personnalisé:
  `$docker network create -d bridge --subnet 172.30.0.0/16 mynetwork`
- Ajouter un conteneur à un réseau:
  `$docker run -tid --name conteneur1 --network mynetwork  alpine`
  ou
  `$docker run -tid --name conteneur1 --net=mynetwork  alpine`
- Autres cas `--net`
  - --net : none
  - --net : host
  - --net : container:<nomconteneur>
  - --link : <nomconteneur>
- ajout dans /etc/hosts du conteneur: --add-host <nomhost>:ip
- --dns : ajoute les ip de serveurs dns

Manipulations

1. Lancez un conteneur nommé contenur1 à partir de l’image alpine

```
$docker run -tid  --name conteneur1 alpine:latest
```

2. Exécutez un shell bash interactif sur le conteneur1.

```
docker exec -ti conteneur1 sh
```

3. Quel est l’adresse IP attribué à votre conteneur1

```
#ip a
[...]
    link/ether 02:42:ac:11:00:08 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.8/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
[…]
```

4. Faites un ping sur la machine 172.17.0.1 (la machine qui héberge Docker).

```
#ping 172.17.0.1
```

5. Quittez le conteneur

```
#exit
```

6. Vérifiez que la machine qui héberge Docker a l’adresse 172.17.0.1.

```
$ip a
[…]
docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
default
    link/ether 02:42:09:ae:c4:ac brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
```

7. Supprimez conteneur1

```
$docker rm -f conteneur1
```

8. Visualisez les réseaux disponible

```
docker network ls
  NETWORK ID          NAME                DRIVER          SCOPE

 7fca4eb8c647        bridge              bridge          local
 9f904ee27bf5        none                null            local
 cf03ee007fb4        host                host            local
```

9. Créez un réseau de type bridge ayant l'IP 172.30.0.0/16 et ayant monreseau comme nom

```
$docker network create -d bridge --subnet 172.30.0.0/16 monreseau
```

10. Affichez les détails de monreseau

```
$docker network inspect monreseau
```

11. Lancez un conteneur nommé conteneur1 à partir de l'image alpine et dont l'IP appartient au réseau monreseau

```
$docker run -tid  --name conteneur1  --network monreseau alpine:latest
```

12. Exécutez un shell bash interactif sur le conteneur1.

```
$docker exec -ti conteneur1 sh
```

13. Vérifiez que l'adresse IP de conteneur1 est dans le réseau 172.30.0.0/16

```
#ip a
```

14. Quittez le shell qui s'exécute sur conteneur1

```
#exit
```

15. Lancez un conteneur nommé conteneur2 à partir de l'image alpine et dont l'IP appartient au réseau monreseau

```
$docker run -tid  --name conteneur1  --network monreseau alpine:latest
```

16. Exécutez un shell bash interactif sur le conteneur2.

```
$docker exec -ti conteneur2 sh
```

17. Vérifiez que l'adresse IP de conteneur2 est dans le réseau 172.30.0.0/16 puis faites un ping sur conteneur1

```
#ip a
[…]
ping 172.30.0.2
```

18. Quittez le shell qui s'exécute sur conteneur2.

```
#exit
```

19. Supprimez les deux conteneurs.

```
$docker  rm -f conteneur{1,2}
```

20. Lancez un conteneur nommé conteneur1 à partir de l'image alpine.

```
$docker run -tid  --name conteneur1 alpine:latest
```

21. Lancer un conteneur nommé conteneur2 à partir de l'image alpine tout en établissant un lien réseau avec conteneur1.

```
$docker run -tid  --name conteneur2 --link conteneur1 alpine:latest
```

22. exécutez un shell bash interactif sur le conteneur2.

```
$docker exec -ti conteneur2 sh
```

23. Affichez le contenu de /etc/hosts? Que remarquez vous?

- - - -

Activité n°7: Docker et microservices

Activité de lecture

Titre: Construisez des Microservices

URL: *https://openclassrooms.com/fr/courses/4668056-construisez-des-microservices/5122300-apprehendez-larchitecture-microservices*

Exemple minimaliste

- 1 page **index.html** à mettre à jour
- 2 variables calculées(bash)
  ==>
- 3 conteneurs :
  - 1 conteneur exécutant un serveur web Nginx (affichage de la page contenant la valeur des variables)
  - 2 conteneurs php (workers)==>1 par variable

1. Créez l'arborescence suivante

```
myapp
├── sereurweb
├── worker1
└── worker2
```

```
mkdir -p myapp/{sereurweb,worker1,worker2}
tree myapp
```

2. Créez vos Dockerfiles et vos scripts

Dans le répertoire myapp/sereurweb

nano myapp/sereurweb/Dockerfile

```
 FROM ubuntu:latest
RUN apt-get update
RUN apt-get install -y nginx
VOLUME /var/www/html/
ENTRYPOINT ["nginx","-g","daemon off;"]
```

Dans le répertoire myapp/worker1

nano myapp/worker1/Dockerfile

```
FROM php:7.2-cli
COPY rollon.sh /
COPY affichage.php /
RUN chmod 755 /rollon.sh
ENTRYPOINT ["./rollon.sh"]
```

nano myapp/worker1/rollon.sh

```
#!/bin/bash
x=1
```

```
while true
do
echo $x > /var/www/html/worker1.txt
((x=x+100))
php /affichage.php
sleep 10
done
```

nano myapp/worker1/affichage.php

```
<?php
$file1='/var/www/html/worker1.txt';
$file2='/var/www/html/worker2.txt';
$Data1="";
$Data2="";

if(file_exists($file1)){
  $fh = fopen($file1,'r');
  while ($line = fgets($fh)){
    $worker1 = $line;
  }
  fclose($fh);
}

if(file_exists($file2)){
  $fh = fopen($file2,'r');
  while ($line = fgets($fh)){
    $worker2 = $line;
  }
  fclose($fh);
}

$File = "/var/www/html/index.html";
$Handle = fopen($File, 'w');
$Data1 = "worker1 vaut".$worker1."\n";
fwrite($Handle, $Data1);
$Data2 = "worker2 vaut".$worker2."\n";
fwrite($Handle, $Data2);
fclose($Handle);
```

Dans le répertoire myapp/worker2

nano myapp/worker2/Dockerfile

```
FROM php:7.2-cli
COPY rollon.sh /
COPY affichage.php /
RUN chmod 755 /rollon.sh
ENTRYPOINT ["./rollon.sh"]
```

nano myapp/worker2/rollon.sh

```
#!/bin/bash
x=1
while true
do
echo $x > /var/www/html/worker2.txt
((x=x+100))
php /affichage.php
sleep 10
done
```

nano myapp/worker2/affichage.php

```
<?php
$file1='/var/www/html/worker1.txt';
$file2='/var/www/html/worker2.txt';
$Data1="";
$Data2="";

if(file_exists($file1)){
  $fh = fopen($file1,'r');
  while ($line = fgets($fh)){
    $worker1 = $line;
  }
  fclose($fh);
}

if(file_exists($file2)){
  $fh = fopen($file2,'r');
  while ($line = fgets($fh)){
    $worker2 = $line;
  }
  fclose($fh);
}

$File = "/var/www/html/index.html";
$Handle = fopen($File, 'w');
$Data1 = "worker1 vaut".$worker1."\n";
fwrite($Handle, $Data1);
```

```
$Data2 = "worker2 vaut".$worker2."\n";
fwrite($Handle, $Data2);
fclose($Handle);
```

3. Créez vos images

```
$cd myapp/serveurweb
$docker build -t serveurweb .
$cd ../worker1
docker build -t worker1 .
cd ../worker2
$dockerbuild -t worker2 .
```

4. Lancez vos services

```
$docker run -tid  --name site serveurweb
$docker run -tid --name worker1 --volumes-from  site worker1
$docker run -tid --name worker2 --volumes-from  site worker2
```

Activité n°8: Sécuriser le user namespace

Points importants

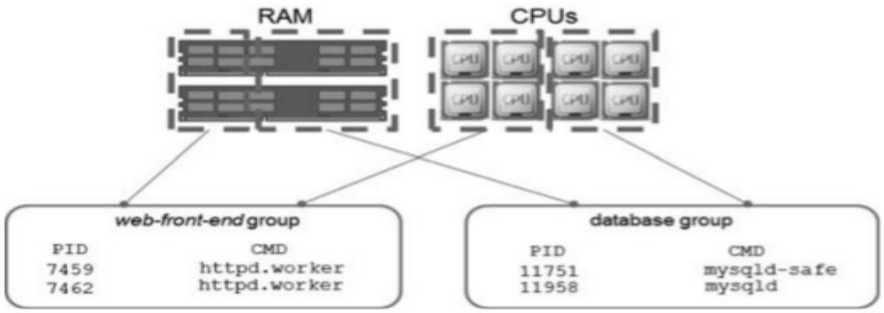
Docker utilise des fonctionnalités du noyau Linux pour assurer l'isolation des containers, telles que les cgroups et les namespaces

CGroups

CGroups (pour Control Groups) permet de partitionner les ressources d'un hôte (processeur, mémoire, accès au réseau ou à d'autres terminaux). L'objectif est de contrôler la consommation de ces ressources par processus.

On doit cette bibliothèque aux ingénieurs de Google, qui a été probablement la première entreprise commerciale à utiliser des conteneurs, bien avant l'apparition de Docker.

Prenons par exemple une machine sur laquelle serait installée une application web avec un front-end PHP et une base de données MySQL. Cgroups permettrait de répartir la puissance de calcul (CPU) et la mémoire disponible entre les différents processus, afin de garantir une bonne répartition de la charge (et donc probablement des temps de réponse).



CGroups introduit une notion de contrôleur qui, comme son nom l'indique, a pour objectif de contrôler l'accès à une ressource pour une hiérarchie de processus. En effet, par défaut, les processus fils d'un processus associé à un groupe donné héritent des paramètres de leur parent.

Namespaces

Les Namespaces sont indépendants de CGroups, mais fonctionnent de concert. Ils permettent de faire en sorte que des processus ne voient pas les ressources utilisées par d'autres. Si CGroups gère la distribution des ressources, Namespaces apporte l'isolation nécessaire à la création de conteneurs.

L'ancêtre du mécanisme des Namespaces est la commande chroot, qui existe au sein du système Unix depuis 1979 ! La fonction de cette

commande est de changer pour un processus donné le répertoire racine du système. Le processus en question a l'impression d'être à la racine du système. L'objectif étant, par exemple, pour des raisons de sécurité, d'empêcher l'utilisateur de se promener dans des sections du système de fichiers dont il n'aurait a priori pas l'usage. Il s'agit donc conceptuellement d'une sorte de virtualisation. Namespaces étend ce concept à d'autres ressources:

- IPC: Communication interprocessus
- Network: Terminaux réseau, ports, etc.
- Mount: Point de montage (système de fichiers)
- PID: Identifiant de processus
- User: Utilisateurs et groupes
- UTS: Nom de domaines

Manipulations

1. Supprimez les conteneurs qui se trouvent sur votre machine.

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

2. Mettez en place un conteneur persistant en exécutant la commande suivante:

```
$docker run -d --name usertest ubuntu:latest sleep infinity
```

3. Vérifiez que votre conteneur est en cours d'exécution

```
$docker ps
```

4. Exécutez la commande ps aux|grep "sleep infinity". Avec quel identité s'exécute le processus que vous avez lancez dans votre conteneur?

---

5. Lisez l'article « Isolate containers with a user namespace » publié dans docker docs et en déduire le rôle de script ci-dessous:

```
#!/bin/bash
groupadd -g 500000 dockremap &&
groupadd -g 501000 dockremap-user &&
useradd -u 500000 -g dockremap -s /bin/false dockremap &&
useradd -u 501000 -g dockremap-user -s /bin/false dockremap-user

echo "dockremap:500000:65536" >> /etc/subuid &&
echo "dockremap:500000:65536" >>/etc/subgid

echo "
  {
   \"userns-remap\": \"default\"
  }
```

```
" > /etc/docker/daemon.json

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

6. Exécutez le script ci-dessus:

```
$ chmod +x userremap.sh
# ./userremap.sh
```

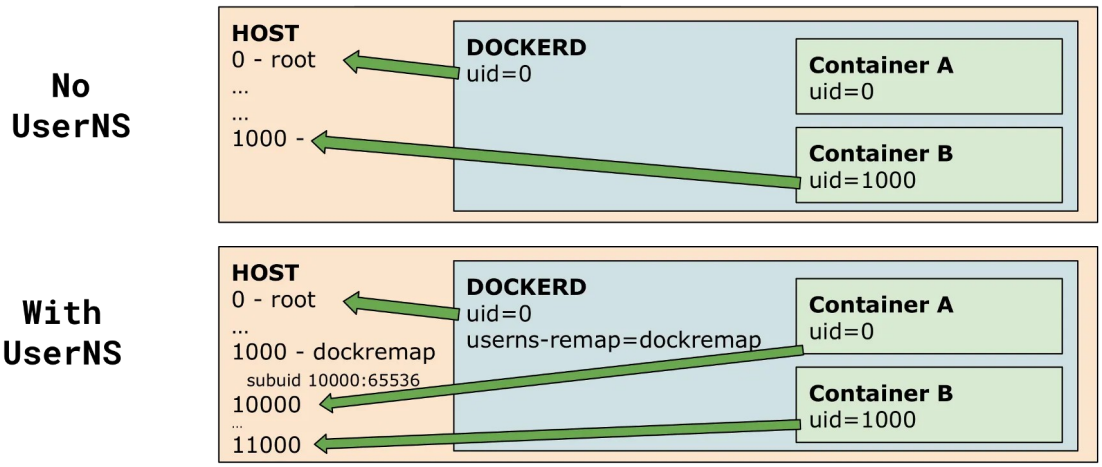
7. Détruisez le conteneur usertest.

```
$docker rm -f usertest
```

8. Refaites les questions 2,3 et 4.

Recap

La fonctionnalité des espaces de noms d'utilisateurs dans Docker consiste à associer les utilisateurs à l'intérieur des conteneurs à une plage différente dans l'hôte, ce qui offre une meilleure sécurité dans le cas où le conteneur a accès aux mêmes ressources externes.



Activité n°9: Registry- tags-pull-push

Points importants

- Les tags
  - très important pour le versionning.
  - docker tag <image_source>:<version> <images_dest>:<version>
- Le push / pull
  - si registry par défaut (docker hub): docker login
  - si registry particulière: docker login <chemin_registry>
  - push: docker push <registry><nom_image>:<version>
  - pull: docker pull <registry><nom_image>:<version>

Manipulations

1. Créez un répertoire tpp

```
$mkdir tpp
```

2. Dans le répertoire tpp, créez un fichier Dockerfile.

```
$cd tpp
$nano Dockerfile
      FROM alpine:latest
      LABEL maintener m2etudiant
```

3. Exécutez la commande suivante: `$docker  build -t myalpine:v1.0 .`? Que réalise-t-elle?

---

4. Exécutez la commande `$docker  image ls` et vérifiez que l'image myalpine figure bien dans la liste des images.

5. Exécutez la commande `$docker  run -it myalpine`. Que se passe-t-il?

---

6. Comment remédier à ce problème?

```
$docker tag myalpine:v1.0  myalpine:latest
```

7. Vérifiez le résultat de votre travail en exécutant `$`.

```
REPOSITORY      TAG               IMAGE ID          CREATED             SIZE
myalpine        V1.0              6b49b0ba1e48      8 minutes ago       5.57MB
myalpine         latest            6b49b0ba1e48       8 minutes ago       5.57MB
[…]
```

8. Exécutez de nouveau la commande `$docker  run -it myalpine`. Que se passe-t-il?

---

9. Remontez vers la version 2.0

```
$docker tag myalpine:v1.0  myalpine:v2.0
$docker tag myalpine:v2.0  myalpine:latest
```