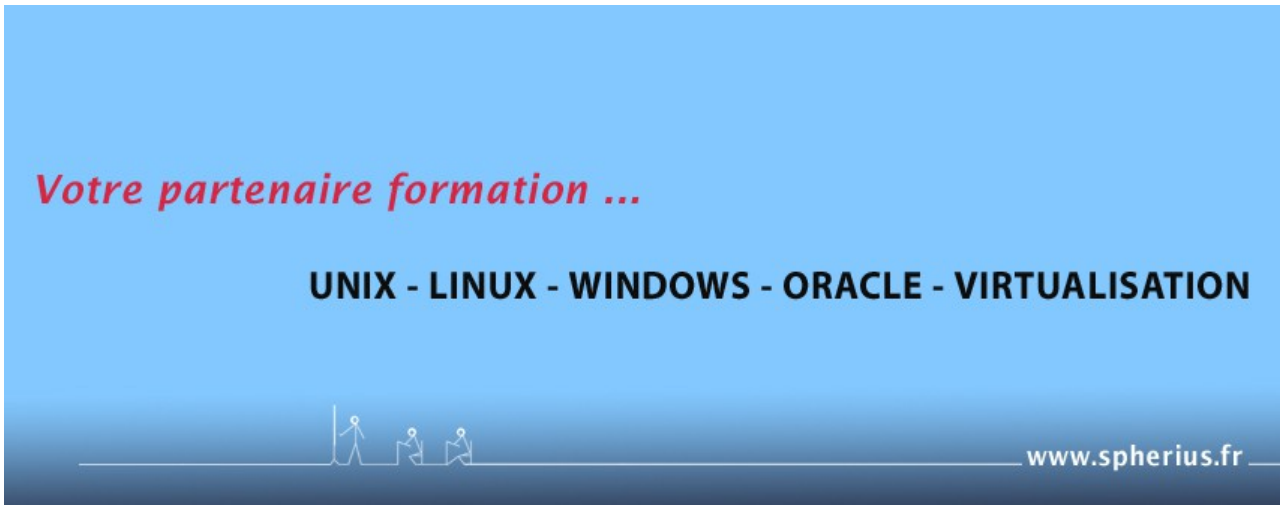
DOCKER

EXERCICES



SOMMAIRE

Les auteurs de ce document sont :

- Monsieur Baranger Jean-Marc,
- Monsieur Schomaker Theo.

Installation et configuration de docker

Exercice 1 : L'installation de Docker

Installer le dépôt docker.
Mette à jour les méta-données.
Installer Docker.
Démarrer Docker et l'activer pour le prochain redémarrage.
Afficher le statut du service Docker.
Afficher la version de Docker de manière succincte et détaillée.
Vérifier qu'un groupe appelé docker a été créé.
Vérifier qu'une interface réseau docker0 a été créé.

Exercice 2 : L'aide de docker

Afficher l'aide de la commande docker de deux manières différentes.
Afficher l'aide de la sous-commande 'search' de deux manières différentes.
Afficher l'aide de la sous-commande 'image' de deux manières différentes.
Afficher l'aide de la sous-commande 'run' de deux manières différentes.
Afficher l'aide de la sous-commande 'volume' de deux manières différentes.
Afficher l'aide de 'create' de la sous-commande 'volume' de deux manières différentes.

Docker en ligne de commandes

Préalable : exécuter les commandes suivantes :

```
# docker rm -f $(docker ps -qa)
# docker rmi $(docker images -q)
```

Exercice 1 : Commandes de recherche

Rechercher les images ubuntu.
Rechercher les images ubuntu qui ont plus de 50 étoiles.
Rechercher les images ubuntu en n'affichant que 5 résultats.
Du résultat précédent, afficher en entier le champ de description.
Rechercher les images ubuntu qui sont officielles.

Exercice 2 : Exécution d'un conteneur

Exécuter (run) un conteneur ubuntu en affichant le fichier /etc/passwd.
Que s'est il passé ?

Recommencer en affichant le fichier /etc/hosts.
Quelle est la différence avec l'étape numéro 1 ?

Combien de conteneurs sont présents ?
Combien d'images sont présentes dans le cache local ?

Exécuter un conteneur nommé dock1 d'une image debian en mode démon (arrière plan) et interactif.

Lister les conteneurs, ainsi que ceux en cours de fonctionnement.

Arrêter les conteneurs et les supprimer via la commande suivante :

```
# docker stop  nom_conteneur_1  nom_conteneur_2  nom_conteneur_3 . . .
# docker rm    nom_conteneur_1  nom_conteneur_2  nom_conteneur_3 . . .
```

Create / pull / start / exec

Lister les images nginx pour vérifier qu'il n'y en a pas de présente.

Transférer l'image nginx (la dernière version) dans le cache local de votre poste. Vérifier.

Créer un conteneur de l'image nginx sans le démarrer et nommé web1.
Lister les conteneurs (vérifier le status de web1).

Démarrer le conteneur web1.
Lister les conteneurs, ainsi que ceux en cours de fonctionnement.

Se connecter au conteneur web1 via la sous commande 'exec', les options -it et la commande /bin/bash.
Une fois connecté, taper les commandes hostname et 'ls /usr/share/nginx/html'.
Quitter le conteneur sans l'arrêter. Vérifier qu'il soit toujours en cours de fonctionnement.

Arrêter le conteneur web1 et le supprimer.

run / option dit et it

Exécuter un conteneur nommé cont1 d'une image debian en mode démon (arrière plan), interactif et en associant un terminal : -dit
Lister les conteneurs en cours de fonctionnement.
Exécuter un conteneur nommé cont2 d'une image debian en interactif et en associant un terminal : -it
Que constatez-vous ?
Reprendre la main avec 'CTRL + PQ'.
Lister les conteneurs en cours de fonctionnement. Le conteneur cont2 est-il arrêté ?

Utiliser la sous commande attach pour se reconnecter au conteneur cont2.
Quitter avec 'CTRL + PQ'. Lister les conteneurs en cours de fonctionnement. Que constatez-vous ?
Utiliser la sous commande attach pour se reconnecter au conteneur cont2.
Quitter avec 'exit'. Lister les conteneurs en cours de fonctionnement. Que constatez-vous ?

Supprimer le conteneur cont1, alors qu'il en cours de fonctionnement (sans utiliser la sous commande stop).
Supprimer le conteneur cont2.

Exécuter la commande suivante : docker run -it --name cont1 debian ping 8.8.8.8
Faire 'CTRL + C'
Exécuter la commande suivante : docker run -dit --name cont2 debian ping 8.8.8.8
Lister les conteneurs.
Se connecter au conteneur cont2 par : docker attach cont2
Quitter avec 'CTRL + PQ'. Lister les conteneurs en cours de fonctionnement. Que constatez-vous ?
Utiliser la sous commande attach pour se reconnecter au conteneur cont2.
Quitter avec 'CTRL + C'. Lister les conteneurs en cours de fonctionnement. Que constatez-vous ?

Faire un « grand nettoyage » via : docker system prune

Inspect / logs

Lister le contenu du répertoire /var/lib/docker/containers. Le répertoire est vide.

Exécuter un conteneur nommé monweb d'une image httpd avec les options (-dit)
Noter le début du nom qui s'est affiché comme résultat de la commande.

Lister les conteneurs, et noter le 'container id'.

Inspecter le conteneur.
Remarquer 'ID', 'State'.

Lister le contenu du répertoire /var/lib/docker/containers.

Exécuter la commande suivante : docker inspect -f '{{.Config.Hostname}}' monweb

Consulter la configuration réseau du conteneur monweb via 'docker inspect'.
Noter l'adresse IP et la gateway.
Vérifier son bon fonctionnement par : curl http://adresse_ip_conteneur_monweb

Renommer le conteneur monweb en monsite.
Vérifier.

Générer de l'activité par : curl http://adresse_ip_conteneur_monsite
Afficher les logs de monsite.
Générer à nouveau de l'activité par :curl http://adresse_ip_conteneur_monsite
Afficher les logs de monsite.

Affichez les statistiques d'utilisation des ressources de monsite.

Arrêter monsite.

Exécuter un conteneur avec les caractéristiques suivantes :
nom : perso
de l'image debian
en mode démon, interactif et en affectant un speudo terminal
le conteneur se supprimera à son arrêt.
Il doit exécuter la commande : ping 8.8.8.8

Lister les conteneurs en cours de fonctionnement.

Utiliser les sous commandes top, puis stats, puis logs pour le conteneur 'perso'.

Afficher la différence entre le conteneur et son image. A ce stade, il ne doit rien afficher.

Se connecter au conteneur perso via : docker exec -it perso /bin/bash
Créer un fichier fic et supprimer /opt (touch /fic ; rm -rf /opt).
Quitter par CRL+PQ.
Afficher la différence entre le conteneur et son image.

En une commande, supprimer tous les conteneurs sans préciser le nom des conteneurs.

Exercice 3 : Les variables

Rechercher les images mariadb.
Télécharger l'image de mariadb.

Exécuter en arrière plan un conteneur nommé sql0 de l'image mariadb.
Constater que le conteneur a été créé mais ne fonctionne pas. Démarrer le conteneur et constater qu'il est arrêté.

Exécuter en arrière plan un conteneur nommé sql1 de l'image mariadb avec les variables suivantes :
MYSQL_USER=user1
MYSQL_PASSWORD=mypass75
MYSQL_DATABASE=db1
MYSQL_ROOT_PASSWORD=mypass75

Vérifier que le conteneur s'exécute bien.
Afficher l'adresse IP du conteneur.

Installer la commande mysql.

```
# yum install -y mariadb
```

Connectez-vous à la base de données, puis exécuter les commandes suivantes : help, status, quit.

```
#  mysql -u user1 -h adresse_ip -p db1
```

Créer un nouveau container appelé sql2 en regroupant les variables dans un fichier.
Connectez-vous à la base de données, puis exécuter les commandes suivantes : help, status, quit.

Supprimer les trois conteneurs.

Exercice pour le hostname

Démarrer un conteneur en arrière plan, nommé site1, de l'image nginx.
Démarrer un conteneur en arrière plan, nommé site2, de l'image nginx avec le hostname à client2.
Vérifier leurs présences en cours d'exécution.

Afficher les adresses IP des deux conteneurs.
Afficher le hostname des deux conteneurs.

Supprimer les deux conteneurs.

Exercice 4 : Les copies, pause, unpause et wait

Préalable : le fichier index_base.html est fourni et il est localisé dans votre répertoire 'sources'.

Démarrer un conteneur en arrière plan, nommé site1, de l'image nginx.
Démarrer un conteneur en arrière plan, nommé site2, de l'image nginx.

Vérifier le bon fonctionnement en consultant la page de garde des deux conteneurs.
curl http://adresse_ip_du_conteneur
ou via votre browser http://adresse_ip_du_conteneur

Copier le fichier index_base.html de votre répertoire 'sources' au sein du conteneur site1 (le répertoire est /usr/share/nginx/html et le fichier est nommé index.html). Consulter la page de site.

Essayer de copier le fichier index.html de votre conteneur site1 au sein du même répertoire du conteneur site2. Que constatez-vous ?

Mettre en pause le conteneur site1. Consulter l'état par 'docker ps'. Consulter la page de site.

Retirer le conteneur site1 du mode pause. Consulter l'état par 'docker ps' et la page de site.

Arrêter le conteneur site2.

Ouvrir une nouvelle fenêtre terminale et taper :

```
# echo "debut de mon script" ; docker wait site1 ; echo "suite de mon script" ; docker start site2
```

Que fait cette séquence de commandes ?

De la première fenêtre terminale, vérifier l'état des conteneurs par 'docker ps -a'. Puis mettre en pause le conteneur site1. Que constatez-vous dans l'autre fenêtre terminale ?
vérifier l'état des conteneurs par 'docker ps -a'.
Retirer le conteneur site1 du mode pause. Que constatez-vous dans l'autre fenêtre terminale ?
vérifier l'état des conteneurs par 'docker ps -a'.
Arrêter le conteneur site1. Que constatez-vous dans l'autre fenêtre terminale ?
vérifier l'état des conteneurs par 'docker ps -a'.

Supprimer les deux conteneurs.

Exercice 5 : Arrêt et démarrage d'un conteneur

Créer quatre conteneurs web d'après l'image de nginx appelés respectivement web_no, web_always, web_unless_stropped et web_on_failure. Utiliser pour chacun des conteneurs la valeur adéquate de l'option 'restart' en fonction de leur nom.
Faites les fonctionner en mode démon.

Vérifier que les quatre containers s'exécutent.

Exécuter la commande 'pkill -9 nginx'.
Quels conteneurs sont redémarrés et ceux qui sont arrêtés ?

Redémarrer le(s) conteneur(s) arrêté(s).

Redémarrer le service docker. Vérifier quels conteneurs ont redémarré et lesquels ont été arrêté.
Quels conteneurs sont redémarrés et ceux qui sont arrêtés ?

Arrêter les conteneurs qui sont démarrés.

Redémarrer le service docker (équivalent à un redémarrage de votre serveur).

```
# systemctl restart docker
```

Quels conteneurs sont redémarrés et ceux qui sont arrêtés ?

Supprimer les conteneurs de cet exercice.

Exercice 6 : Les volumes

Préalable : dans votre répertoire 'sources', les fichiers base.connec et annuaire.sql sont fournis.

Faire un « grand nettoyage » via :

```
# docker  system prune --volumes
```

Vérifier que le répertoire /var/local est vide.

Exécuter un conteneur en mode démon nommé bdd1 de l'image mariadb avec un volume persistant. Pour ce volume, les données de mariadb (répertoire /var/lib/mysql) devront être accessibles depuis le répertoire /var/local/mariadb de l'hôte.
Vous utiliserez également les variables du fichier base.connec.

Vérifiez que le container fonctionne et consulter le répertoire /var/local.

Exécuter la commande suivante pour créer une table 'annuaire' dans la base de données :

```
# mysql -u user1 -h 172.17.0.2 -p db1 < ~/Sources/annuaire.sql
   saisir le mot de passe
```

Vérification par :

```
# mysql -u user1 -h 172.17.0.2 -p db1
   saisir le mot de passe
   select * from annuaire ;
   quit
```

Connectez-vous au conteneur (via la commande docker exec).
Au sein du conteneur, créer un fichier quelconque (exemple : TEST.DB) dans /var/lib/mysql.
Déconnectez-vous du conteneur.
De votre machine hôte, le fichier apparaît-il dans /var/local/mariadb ?

Arrêter et supprimer le conteneur.

Essayer d'accéder à la base de donnée avec la commande suivante, vous aurez un échec.

```
# mysql -u user1 -h 172.17.0.2 -p db1
```

Le répertoire /var/local/mariadb est-il présent ? Si oui, le fichier créé existe-t-il toujours ?

Exécuter un nouveau conteneur en mode démon nommé autrebdd de l'image mariadb avec le même partage de volumes et avec le fichier base.connec.

Connectez-vous au conteneur (via la commande docker exec).
Au sein du conteneur, le fichier créé précédemment dans /var/lib/mysql est-il présent ?
Quitter le conteneur.

Exécuter :

```
# mysql -u user1 -h 172.17.0.2 -p db1
   saisir le mot de passe
   select * from annuaire ;
   quit
```

Qu'en déduisez-vous ?

Lister les volumes par 'docker volume ls'.

Afficher les informations détaillées de votre conteneur pour retrouver les informations du volume.

Supprimer les conteneurs de cet exercice.

Exercice 7 : Les ports réseaux

Préalable : les fichiers index_base.html et index_autre.html sont dans votre répertoire 'sources'.

Dans votre répertoire de connexion (/root), créer deux répertoires rep_a et rep_b.
Copier dans les deux répertoires le fichier index_base.html en le renommant index.html.

Exécuter 4 conteneurs de l'image nginx :
- site2 : en mappant le port 92 du poste local au port 80 du conteneur,
  en mappant le répertoire rep_a du poste local avec /usr/share/nginx/html du conteneur.
- site3 : en mappant le port 93 du poste local au port 80 du conteneur,
  en mappant le répertoire rep_a du poste local avec /usr/share/nginx/html du conteneur.
- site4 : en mappant le port 94 du poste local au port 80 du conteneur,
  en mappant le port 194 du poste local au port 443 du conteneur,
  en mappant le répertoire rep_b du poste local avec /usr/share/nginx/html du conteneur.
- site5 : en mappant le port 95 du poste local au port 80 du conteneur,
  en mappant le port 195 du poste local au port 443 du conteneur,
  en mappant le répertoire rep_b du poste local avec /usr/share/nginx/html du conteneur.

Lister les conteneurs en cours de fonctionnement, consulter la section PORTS.
Lister les ports du conteneur site5
Inspecter en détail le conteneur site5 pour retrouver les informations du mappage des ports.

Afficher la page de garde de chaque conteneur via http://localhost:port.
Modifier le fichier index.html du répertoire rep_b en récupérant le fichier index_autre.html.
Afficher la page de garde de chaque conteneur via http://localhost:port.

Redémarrer le conteneur site5. Lister les ports.

Exécuter un conteneur site6 de l'image nginx en mappant les ports de nginx aléatoirement avec les ports du poste hôte.
Afficher les ports de site6.

Supprimer les conteneurs de cet exercice.

Exercice 8 : Gestion des ressources

Exercice 7.1 : Gestion des ressources CPU

Créer trois conteneurs basés sur une Débian appelés respectivement deb1, deb2 et deb3 avec les contraintes suivantes :

Affecter 10 parts de CPU à deb1
Affecter 30 parts de CPU à deb2
Affecter 60 parts de CPU à deb3

Connectez-vous aux conteneurs. Exécutez la commande 'yes > /dev/null &' dans chacun des conteneurs. Cette commande va générer une saturation de l'utilisation CPU.

Sur la machine hôte afficher le taux d'utilisation CPU (commande top).
Vérifier que deb1, deb2 et deb3 utilisent respectivement 10% , 30% et 60% des ressources CPU.

Arrêter le conteneur deb3. Sur la machine hôte afficher le taux d'utilisation CPU, vérifier que la répartition de charge respecte les parts.

Supprimer les 3 conteneurs.

Exercice 7.2 : Gestion des ressources mémoire

Exécuter la commande 'free -h' pour constater la mémoire totale du système hôte.

Démarrer un conteneur nommé cent1 d'une image CentOS avec une limitation mémoire à 100 mégas.

Au sein de ce conteneur :

- installer les packages epel-release, puis stress.
- exécuter la commande 'free -h' pour vérifier l'utilisation mémoire et la mémoire totale.
- générer une utilisation mémoire de 100 mégas par la commande :
  stress --vm 1 --vm-bytes 100M &
- vérifier l'utilisation mémoire de la commande précédente :
  ps -aux la colonne %MEM par rapport à la mémoire totale
- arrêter la commande stress : pkill stress
- générer une utilisation mémoire de 200 mégas par la commande :
  stress --vm 1 --vm-bytes 200M &
- vérifier l'utilisation mémoire de la commande précédente :
  ps -aux la colonne %MEM par rapport à la mémoire totale
  Que constatez-vous ?
- arrêter la commande stress : pkill stress

Supprimer le conteneur.

Création d'une image personnalisée

Préalable :
Arrêter tous les conteneurs et les supprimer, ainsi que les volumes.

```
# docker stop $(docker ps -q)
# docker system prune --volumes
```

Exercice 1 : Recherche de version d'images

Nota : le nom du registry est : registry.hub.doker.com

Afficher tous les tags de pour l'image centos
Afficher tous les tags de pour l'image mariadb

Exercice 2 : Création d'une image à partir d'un container

Exécuter un container ubuntu appelé ubu1 et connectez-vous dedans.

Créer les utilisateurs user1 et user2 avec leur répertoire de connexion.

```
root@b3c96210418d:/# useradd -m user1
root@b3c96210418d:/# useradd -m user2
```

Sortir du conteneur et créer une image appelée monimage à partir de celui-ci.

Lister l'image. Afficher les informations détaillées de cette image.
Afficher l'historique de l'image monimage (la sous commande history).
Consulter les répertoires /var/lib/docker, /var/lib/docker/containers, /var/lib/docker/image, ...

Analyser les résultats de inspect (champs ID, Parent, Container, Hostname, …) avec ceux de history (colonne IMAGE) et de l'arborescence /var/lib/docker.

Créer un nouveau conteneur appelé ubu2 à partir de monimage.

Les utilisateurs créés dans ubu1 sont-ils présents au sein de ubu2 ?

Sortir du conteneur ubu2, lister les conteneurs. Pour ubu1 et ubu2, noter les container_id.
Consulter le contenu de /var/lib/docker/containers.

Créer le tag 1.0 pour l'image monimage. Lister les images monimage.
Créer une image monimage:2.0 à partir de ubu2. Au préalable arrêter ubu2. Lister les images.
Supprimer l'image monimage:latest. Lister les images monimage.
Créer le tag latest pour l'image 2.0. Lister les images monimage.

Utiliser la sous commande history sur l'image 1.0 puis sur l'image 2.0. Que constatez-vous ?

Lister le contenu de /var/lib/docker/containers.
Supprimer les deux conteneurs ubu1 et ubu2.
Lister le contenu de /var/lib/docker/containers.

Supprimer les images monimage.

Exercice 3 : Création d'une image personnalisée

Préalable : le fichier Dockerfile est fourni dans le répertoire de l'exercice.

Soit le fichier Dockerfile suivant.

```
# more Dockerfile
FROM centos:latest
RUN yum update -y
RUN yum install -y wget gcc make
RUN wget https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
RUN tar zxf hello-2.10.tar.gz
RUN cd hello-2.10  && ./configure && make && make install
RUN yum remove -y wget gcc make
```

Analyser le fichier Dockerfile.
Créer une image appelée testimg1 à partir de celui-ci.
Afficher la taille de l'image.
Exécuter un conteneur pour vérifier que la commande hello fonctionne. Utiliser l'option supprimant le conteneur à son arrêt.

Syntaxe de la commande : hello [-g autre_message]

Exercice 4 : QCM d'optimisation de l'image

Sélectionner les possibilités qui permettent d'optimiser l'image précédente :

- supprimer la ligne '`RUN yum install -y wget gcc make`' car les packages sont supprimées à la fin. On peut donc retirer également la ligne '`RUN yum remove -y wget gcc make`'.
- utiliser un seul RUN et séquencer les commandes avec des &&.
- utiliser 3 RUN. Le 1er pour le 'yum update' et le 'yum install'. Le 2eme pour les trois RUN suivants et le 3eme pour le dernier RUN.
- ne pas utiliser l'image centos mais 'FROM /bin/bash'.
- ne pas utiliser l'image centos mais plutôt l'image alpine.
- ajouter une ligne RUN en fin de fichier pour retirer également les mans.
- avoir un FROM centos (ou alpine) avec un volume pour pointer sur le répertoire du hôte où est localisée la commande hello.

Exercice 5 : Optimisation de l'image

Préalable : le fichier Dockerfile est fourni dans le répertoire de l'exercice.

Le fichier Dockerfile est la même version que précédemment. Au lieu d'utiliser une image CentOS, il est choisi une image Alpine.

Créer une image appelée testimg:2.0 à partir de ce fichier Dockerfile.
Vérifier la taille de l'image créée, la comparer à la taille de l'image 1.0.

Afficher l'historique de création de l'image.

Modifier le fichier Dockerfile pour créer l'image la plus petite possible appelée testimg:3.0

Créer l'image testimg:3.0 à partir du nouveau fichier Dockerfile. Remarque : l'option -f de 'docker build' permet d'indiquer un fichier différent de Dockerfile pour la construction de l'image.
Vérifier la taille de l'image créée, la comparer aux tailles des images précédentes.

Afficher l'historique de création de l'image .
Qu'en déduisez vous ?

Supprimer les images testimg.

Exercice 6 : Création d'une image à partir de rien (from scratch)

Préalable : le binaire salut est fourni dans le répertoire de l'exercice.

Vérifier le fonctionnement du binaire salut fourni.

```
# ./salut
```

Créer une image appelé 'light' à partir de rien (from scratch).
Pour cela, le Dockerfile est constitué avec :

- le mot clé scratch à la place du nom de l'image
- l'ajout (ADD) du binaire à la racine
- l'exécution de la commande par /salut avec CMD

Afficher la taille de l'exécutable salut.
Afficher la taille de l'image.
Qu'en déduisez-vous ?

Tester cette image en exécutant un conteneur :

```
# docker run -it --rm light
```

Supprimer l'image light.

Exercice 7 : Utilisation du Dockerfile

Préalable : les fichiers index_base.html et index_autre.html sont fournis dans le répertoire de l'exercice.

Préparer un fichier Dockerfile pour créer une image avec les informations suivantes :

- L'image sera la version latest de nginx
- On souhaite intégrer les informations suivantes :
  - maintainer avec votre nom
  - societe avec le nom de votre entreprise
  - version avec la version de votre fichier (exemple : 1.0)
  - description pour une présentation (exemple : Ceci est un exemple)
- Deux variables qui devront être exploitables au sein d'un conteneur :
  - ville avec le nom d'une ville de votre choix
  - pays avec France
- Une variable qui doit être utilisable lors de la création de l'image mais pas dans un conteneur : planete avec une planète de votre choix
- Copier le fichier index_base.html dans l'image à /usr/share/nginx/html/index.html
- Créer un répertoire /opt/contenu_de_la_variable_planete
- Au sein du répertoire créé ci-dessus, créer un fichier vide nommé avec le contenu de la variable pays
- Exposer les ports 80 et 443
- Créer deux volumes pour /usr/share/nginx/html et /var/log/nginx
- le répertoire de travail doit être positionné à /usr/share/nginx

Créer l'image nommée site à partir de votre fichier Dockerfile.

Consulter les informations détaillées de l'image.

Lister le contenu du répertoire /var/lib/docker/volumes.
Exécuter un conteneur en mode démon, nommé web, de cette image.
Lister le contenu du répertoire /var/lib/docker/volumes .

Lister le contenu des deux répertoires /var/lib/docker/containers/*/_data

Faire un 'docker ps'.
Consulter les informations détaillées du conteneur web.

Consulter la page du site.

```
# curl http://ip_du_conteneur
```

Se connecter au conteneur avec 'docker exec -it web /bin/bash', puis effectuer les contrôles suivants :

```
# pwd
# ls -Rl /opt
# echo $ville
# echo $pays
# echo $planete
```

Quitter le conteneur.

```
# exit
```

Supprimer le conteneur, les volumes et l'image.

Exercice 8 : Utilisation du cache

Préalable : le fichier est fourni au sein du répertoire de l'exercice.

Se positionner dans le répertoire de l'exercice.
Consulter le fichier Dockerfile.

Vérifier que vous n'avez pas d'image centos, si c'est le cas la supprimer.

Créer une image test en utilisant la commande suivante (time permet d'afficher le temps total d'exécution de la commande) :

```
# time docker image build -t test .
```

Noter le temps d'exécution.

Recréer l'image de la même manière et noter le temps d'exécution. Que remarquez-vous ?

```
# time docker image build -t test .
```

Recréer l'image de la même manière avec l'option --no-cache et noter le temps d'exécution.
Que remarquez-vous ?

```
# time docker image build -t test --no-cache .
```

Ne pas supprimer l'image 'test'.

Exercice 9 : Sauvegarde / Restauration d'une image

Vérifier la présence de l'image test.

Sauvegarder l'image test dans un fichier monimage.save.

Supprimer l'image test.

Restaurer l'image test.

Vérifier.

Supprimer le fichier de sauvegarde et l'image.

Docker et le réseau

Exercice 1 : Manipulation de réseaux

Créer 2 réseaux de type bridge nommés reseau1 et reseau2.
Lister les réseaux Docker et inspecter les 2 réseaux créés.

Exécuter deux conteneurs de l'image alpine en mode démon, nommés alp1 et alp2, et connectés au réseau 'reseau1'.
Exécuter un conteneur de l'image alpine en mode démon nommé alp3, connecté au réseau 'reseau2' et son hostname sera alpine3.
Inspecter la configuration réseau de ces conteneurs, et noter leurs adresses IP.

```
Remarque pour les adresses IP :
# docker inspect -f '{{.NetworkSettings.Networks.reseau1.IPAddress}}' alp1 alp2
```

Se connecter au conteneur alp1 et analyser sa configuration réseau avec entre autres les commandes suivantes : hostname ; ip a ; ip route et cat /etc/hosts.
Pinguez-vous le conteneur alp2 et alp3, un site externe (8.8.8.8 et www.google.fr), le hôte ?
Se déconnecter du conteneur avec 'ctrl+PQ'.

Consulter la configuration des cartes réseaux de votre hôte (ip a).

Connecter le conteneur alp1 au réseau 'reseau2'.
Inspecter la configuration réseau du conteneur alp1.
Se connecter au conteneur alp1 et analyser sa configuration réseau avec entre autres les commandes suivantes : hostname ; ip a ; ip route, cat /etc/hosts et cat /etc/resolv.conf.
Que constatez-vous ?
Pinguez-vous le conteneur alp3 ?
Se déconnecter du conteneur avec 'ctrl+PQ'.

Inspecter les réseaux 'reseau1' et 'reseau2'. Que constatez-vous ?

Déconnecter le réseau 'reseau2' du conteneur alp1.
Se connecter au conteneur alp1 et exécuter la commande 'ip a'.
Pinguez-vous le conteneur alp3 ?
Se déconnecter du conteneur avec 'ctrl+PQ'.

Supprimer les conteneurs.

Exercice 2 : Configuration DNS

Exécuter un conteneur de l'image alpine en mode démon nommé alp1, sans spécifier de connexion à un réseau et son hostname sera alpine1.
Exécuter un conteneur de l'image alpine en mode démon nommé alp2, connecté au réseau 'reseau1' et son hostname sera alpine2.
Inspecter la configuration réseau de ces conteneurs.

Consulter le fichier /etc/resolv.conf du hôte et des deux conteneurs.

Renommer le fichier /etc/resolv.conf du hôte en /etc/resolv.conf.old
Exécuter un conteneur de l'image alpine en mode démon nommé alp3, sans spécifier de connexion à un réseau.
Exécuter un conteneur de l'image alpine en mode démon nommé alp4, sans spécifier de connexion à un réseau et en spécifiant l'adresse ip 10.10.10.10 comme serveur dns (option '--dns').
Consulter le fichier /etc/resolv.conf des quatre conteneurs.

Renommer le fichier /etc/resolv.conf.old du hôte en /etc/resolv.conf
Supprimer les conteneurs.

Exercice 3 : Configuration customisée d'un réseau

Créer un réseau avec les spécifications suivantes :

- nom : reseau3
- driver : bridge
- sous réseau 192.168.100.0
- netmask : 255.255.255.0 (/24)
- gateway : 192.168.100.254
- ce réseau devra pouvoir avoir 7 adresses IP de 1 à 7 (192.168.100.0/29)

Lister les réseaux et inspecter le réseau 'reseau3'.

Exécuter un conteneur de l'image nginx en mode démon nommé web1 et connecté au réseau 'reseau3'. Consulter sa configuration réseau et son adresse IP.

Exécuter six autres conteneurs de l'image nginx en mode démon nommés de web2 à web7 et connectés au réseau 'reseau3'.

Consulter les adresses IP de ces conteneurs.

```
# docker inspect -f '{{.NetworkSettings.Networks.reseau3.IPAddress}}' \
                  web1 web2 web3 web4 web5 web6 web7
```

Exécuter un autres conteneur de l'image nginx en mode démon nommés de web8 et connecté au réseau 'reseau3'.
Que constatez-vous ?
Vérifier avec 'docker ps', 'docker ps -a', 'docker inspect web8' et 'docker inspect reseau3'.

Exercice 4 : Suppression

Supprimer le réseau 'reseau3'.
Que constatez-vous ?

Arrêter tous les conteneurs nginx.

Vérifier qu'il n'y a plus de conteneurs associés au réseau 'reseau3'.

Utiliser le mot clé 'prune' de la commande 'docker network' pour supprimer tous les réseaux qui ne sont pas utilisés.

Supprimer les conteneurs alpine (nommés alpX) et supprimer les réseaux créés sur ce module ('reseau1', 'reseau2' et 'reseau3').

Docker et le stockage

Exercice 1 : Utilisation d'un volume

Préalable : le fichier /dev/log est un fichier spécial utilisé par le mécanisme syslog.

Objectif de l'exercice : on va associer le fichier spécial /dev/log du hôte à celui d'un conteneur. Ainsi en envoyant un message au mécanisme syslog du conteneur (via la commande logger), le syslog du hôte va les recevoir et les traiter en les plaçant dans ses fichiers de logs.

Créer un conteneur nommé cent1 basé sur une CentOS qui partage le fichier /dev/log avec le fichier /dev/log/ de la machine hôte. Le conteneur devra s'exécuter en mode interactif.

Depuis le conteneur avec la commande logger, envoyer un message à syslogd.

```
[root@94741b841502 /]# logger -p daemon.info "Test Log"
[root@94741b841502 /]# logger -p authpriv.info "Test Log securite"
```

Vérifier l'absence des fichiers /var/log/messages et /var/log/secure au sein du conteneur.
Quitter le conteneur avec 'CTRL+PQ'.

Vérifier que sur la machine hôte vous avez accès aux logs du conteneur et consulter la dernière lignes de ces fichiers.

```
# tail -1 /var/log/messages
# tail -1 /var/log/secure
```

Inspecter le conteneur pour consulter la section 'Mounts'.

Supprimer le conteneur cent1.

Exercice 2 : Création d'un conteneur et d'un volume en lecture seule

Exécuter un conteneur en mode interactif nommé cent1 d'une image centos en partageant le répertoire /data du hôte avec /data du conteneur.
A noter que le répertoire /data du hôte n'existe pas.

Vérifier que le volume est monté.

```
[root@1c0b0bc57e06 /]# df -h
```

Créer un fichier vide fic1 dans / et /data.

```
[root@1c0b0bc57e06 /]# touch /fic1
[root@1c0b0bc57e06 /]# touch /data/fic1
```

Quittez le container avec 'exit'. Supprimer le conteneur cent1.
Vérifier que le fichier fic1 est présent sous /data du hôte.

Exécuter un conteneur en mode interactif nommé cent1 d'une image centos en utilisant l'option --read-only et en partageant le répertoire /data du hôte avec /data du conteneur.
Créer un fichier vide fic2 dans / et /data. Que constatez-vous ?
Le répertoire /data est-il en lecture seule dans le container ?
Quittez le container avec 'exit'. Supprimer le conteneur cent1.
Vérifier que le fichier fic2 est présent sous /data du hôte.

Exécuter un conteneur en mode interactif nommé cent1 d'une image centos en partageant le répertoire /data du hôte avec /data du conteneur en lecture seule (l'option :ro au nom du volume du conteneur).
Créer un fichier vide fic3 dans / et /data. Que constatez-vous ?
Le répertoire /data est-il en lecture seule dans le container ?
Quittez le container avec 'exit'. Supprimer le conteneur cent1.
Vérifier le contenu de /data du hôte.

Supprimer le répertoire /data.

Exercice 3 : Création de volumes

Préalable :
Au du sein du répertoire de l'exercice, le fichier index.html est présent.
Faire du poste hôte un serveur NFS pour partager le répertoire /export/data.

```
# mkdir -p /export/data
# chmod  777  /export/data
# vi /etc/exports
/export/data  *(rw)
# systemctl start nfs
# firewall-cmd --add-service=nfs
# firewall-cmd --add-service=mountd
# firewall-cmd --add-service=rpc-bind
# exportfs
/export/data        <world>
```

Créer un fichier vide fic1 sous /export/data

```
# touch /export/data/fic1
```

Se positionner sur le répertoire de l'exercice et vérifier son contenu.

```
# ls
```

Créer un volume nommé monsite.
Créer un volume nommé meslogs.
Créer un volume nommé sauvegardes correspondant au montage NFS réalisé précédemment.

Lister les volumes. Consulter le détails de ces volumes en les inspectant.

Exécuter un conteneur nommé web1 en mode démon d'une image nginx en mappant le port 91 du hôte avec le port 80 du conteneur et en mappant le volume monsite avec le répertoire /usr/share/nginx/html du conteneur.
Inspecter les informations du conteneur web1 concernant les volumes.

Consulter le site de web1.

```
# curl http://localhost:91
```

Copier le fichier index.html du répertoire de l'exercice au sein du répertoire du volume monsite.

```
# cp index.html /var/lib/docker/volumes/monsite/_data
```

Consulter le site de web1.

Exécuter un nouveau conteneur nommé web2 en mode démon d'une image nginx en mappant le port 92 du hôte avec le port 80 du conteneur et en mappant le volume monsite avec le répertoire /usr/share/nginx/html du conteneur.
Consulter le site.

Supprimer les conteneurs web1 et web2.
Lister le contenu du répertoire du volume monsite.

Exécuter un conteneur nommé web1 en mode démon d'une image nginx en mappant le port 101 du hôte avec le port 80 du conteneur et en mappant le volume monsite avec le répertoire /usr/share/nginx/html du conteneur.
Consulter le site.

Exécuter un conteneur nommé web2 en mode démon d'une image nginx en :

- mappant le port 102 du hôte avec le port 80 du conteneur,
- mappant le volume monsite avec le répertoire /usr/share/nginx/html du conteneur,
- mappant le volume meslogs avec le répertoire /var/log/nginx du conteneur,
- mappant le volume sauvegardes avec le répertoire /backup du conteneur.

Inspecter les informations relatives aux volumes du conteneur web2.

Consulter le site de web2.
Lister le contenu du répertoire /var/lib/docker/volumes/meslogs/_data. Vérifier qu'il y a le même contenu au sein de /var/log/nginx du conteneur web2.
Du conteneur web2, créer un fichier fic2 dans /backup.
Vérifier sur le serveur NFS que le fichier fic2 est présent au sein de /export/data.

En utilisant l'option '--volumes-from', exécuter un conteneur en mode démon nommé web3 d'une image nginx qui utilise les mêmes volumes que le conteneur web2 et en mappant le port 103 du hôte au port 80 du conteneur.

Consulter les volumes de web3 en inspectent les informations de ce conteneur.
Consulter le site de web3
Lister le contenu de /backup de web3.

Supprimer les conteneurs web1, web2 et web.
Supprimer les volumes monsite, meslogs et sauvegardes.

Exercice 4 : Création d'un registry privé

Créer un volume nommé stockage.

Créer un registry privé en exécutant un conteneur en mode démon avec les spécifications suivantes :

- utiliser l'image : registry:2
- nom du conteneur : depot
- mapper le port 5000 du hôte avec le port 5000 du conteneur.
- mapper le volume 'stockage' avec le répertoire /var/lib/registry du conteneur.
- prévoir un démarrage du conteneur avec le démarrage du hôte, ainsi qu'en cas d'arrêt impromptu.

Vérifier le fonctionnement et consulter les informations du conteneur depot.

Le registry privé est prêt à être utilisé. Récupérer une image centos depuis le docker hub et donner lui un tag spécifique pour le registry privé 'depot' (localhost:5000/my_centos).
Vérifier que l'image a bien été taggée.

Copier l'image dans le registry privé 'depot'.

Supprimer les images centos:latest et localhost:5000/my_centos du hôte local.
Vérifier.

Exécuter un conteneur nommé cent1 en mode démon (-dit) à partir de l'image copiée dans le registry privé. Constater que l'image est téléchargée depuis le registry privé.
Lister les images présentes en local et lister les conteneurs en fonctionnement.
L'image est-elle présente?

Supprimer le conteneur cent1 et l'image localhost:5000/my_centos.
Supprimer le conteneur depot.

Recréer un registry privé nommé 'nouveau' avec les mêmes caractéristiques que précédemment.

Exécuter un conteneur nommé cent1 en mode démon (-dit) à partir de l'image copiée dans le registry privé. Constater que l'image est téléchargée depuis le registry privé.
Lister les images présentes en local et lister les conteneurs en fonctionnement.
L'image est-elle présente?

Supprimer les conteneurs cent1 et nouveau.
Supprimer le volume stockage.

Docker Compose

Exercice 1 : Installation de Docker Compose

Afin identifier la version de Docker Compose a installer, consulter le site :
https://github.com/docker/compose/releases

Installer Docker Compose (via curl).
Si la version est 1.23 .0-rc1 :

```
# url_download="https://github.com/docker/compose/releases/download"

# curl -L ${url_download}/1.23.0-rc1/docker-compose-$(uname -s)-$(uname -m) \
       -o /usr/local/bin/docker-compose

# chmod  +x  /usr/local/bin/docker-compose
```

Vérifier par :

```
# docker-compose --version
# docker-compose version
```

Consulter l'aide de la commande :

```
# docker-compose --help
```

Exercice 2 : Un fichier de configuration de base

Se positionner dans le répertoire de l'exercice.
Il s'agit du premier exemple du support de cours. Veuillez à ne pas consulter le support !

Faire l'indentation du fichier docker-compose.yml pour pouvoir l'exécuter.
Rappel : docker-compose config pour valider la configuration du fichier.
docker-compose up -d pour l'exécuter.

Afficher l'aide de la commande docker-compose

Utiliser la commande docker-compose pour :

- lister les images
- lister les services
- afficher le mappage du port 80 du service nginx
- consulter les logs
  - consulter le site : curl http://localhost
  - consulter les logs
- arrêter les services. Vérifier. Redémarrer les services. Vérifier.

Arrêter les services avec suppression des conteneurs, réseaux et volumes les constituants.

```
# docker-compose down
```

Exercice 3 : Le projet wordpress

Se positionner dans le répertoire de l'exercice.
Il s'agit du projet du support de cours.

Exécuter le projet. Corriger l'erreur.
Exécuter à nouveau le projet.

Lister les services.

Arrêter les services avec suppression des conteneurs, réseaux et volumes les constituants.
Redémarrer le projet en lui donnant un nom (utiliser votre prénom).
Lister les services.

Tester en vous connectant à http://localhost:8000 et en créant votre site. Puis connectez-vous.

Utiliser la commande docker pour afficher les réseaux et les volumes.

Inspecter le réseau et le volume du projet.

Utiliser la commande docker-compose pour :

- lister les images
- lister les services
- afficher le mappage du port 80 du service wordpress
- consulter les logs
- utiliser la sous commande top

Ouvrir une autre fenêtre terminale, consulter les événements (vous allez perdre la main).

Revenir à la première fenêtre terminale et générer de l'activité. Consulter à chaque étape le résultat généré sur les events.

- Arrêter le service bdd.
- Lister les services.
- Démarrer le service bdd.

Fermer la fenêtre terminale des events.

Arrêter les services avec suppression des conteneurs, réseaux et volumes les constituants.
Qu'est-ce qui n'a pas été supprimé ?
Vérifier, et tout supprimer.

Exercice 4 : Le projet nginx-php

Se positionner dans le répertoire de l'exercice.
Il s'agit du projet du support de cours.

Exécuter le projet..

Lister les services.

Tester en vous connectant à http://localhost.

Utiliser la commande docker pour afficher les réseaux et les volumes.

Utiliser la commande docker-compose pour :

- lister les images
- lister les services
- consulter les logs
- utiliser la sous commande top

Arrêter les services avec suppression des conteneurs, réseaux et volumes les constituants.
Vérifier.

Exercice 5 : Le projet LEMP

Un projet LEMP qui utilise nginx, php, mariadb et phpmyadmin.

Ce projet permet de développer les relations entre différents conteneurs.
Il met en évidence l'importance de l'organisation d'un projet (arborescence, logs, etc).
Il développe les différentes sous-commandes de docker-compose.

Se positionner dans le répertoire de l'exercice.

Arborescence du projet :

```
# tree  sitelemp
sitelemp/
├── docker-compose.yml
├── logs
│   ├── nginx-access.log
│   └── nginx-error.log
├── nginx
│   └── conf
│       └── default
└── site
    └── index.php
```

Attention lors de vos manipulations, il ne faut pas supprimer les deux fichiers nginx-access.log et nginx-error.log. Si cela arrive, recréez les :

```
# touch logs/nginx-access.log  logs/nginx-error.log
```

Les fichiers du répertoire logs sont des fichiers initialement vide.

```
[root@mars sitelemp]# cat site/index.php
<?php
phpinfo();
```

```
[root@mars sitelemp]# cat nginx/conf/default
server {
  listen 80;
  root /usr/share/nginx/html;
  index index.php index.html index.html;
  server_name 127.0.0.1;
  location / {
    try_files $uri /index.php$is_args$args;
  }
  location ~ \.php$ {
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    fastcgi_pass phpfpm:9000;
    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    include fastcgi_params;
  }
}
```

Le fichier docker-compose n'est pas finalisé.
Éditer ce fichier et compléter le pour ajouter deux services (phpfpm et mariadb) en respectant les spécifications indiquées en commentaires dans ce fichier.

Valider le fichier de configuration (la sous commande config).

Exécuter le projet.

Vérifier le bon fonctionnement :
http://127.0.0.1 on obtient la page d'informations de php (page.php)
http://127.0.0.1:8183 on obtient la page de phpmyadmin

Lister les services.
Inspecter le conteneur sitelemp_web_1_* .
Lister les réseaux avec la commande docker.

Utiliser la sous commande exec de docker-compose pour exécuter sh au conteneur mariadb.
Puis taper les commandes suivantes :

```
mysql  -u  root  -p
        mot de passe : admin
show  databases ;
exit
exit
```

Lister les images.

Consulter les deux fichiers de logs logs/nginx-access.log et logs/nginx-error.log.
Consulter les logs via la commande docker-compose.
Utiliser la sous commande top de docker-compose.

Arrêter le projet.
Vérifier que le site ne fonctionne plus.
Relancer le projet.
Vérifier que le site fonctionne à nouveau.

Arrêter le projet et supprimer les volumes.

Docker Machine

Préalable : Installation de Docker Toolbox.

Du site de Docker, télécharger 'Docker Toolbox' sur votre poste Windows et installer le.
Démarrer 'Docker Quickstart Terminal' pour réaliser les exercices de ce module.

Exercice 1 : Déploiement de machines

Créer quatre machines virtuelles 'Oracle VirtualBox' avec la commande docker-machine.
Elles auront les noms suivants : master, worker1, worker2 et worrker3.

Vérifier qu'elles fonctionnent.

Exercice 2 : Fonctionnalités de docker-machine

Créer une machine virtuelle 'Oracle VirtualBox' via la commande docker-machine nommée 'perso'.

Connectez-vous par ssh à la machine 'perso'et taper les commandes suivantes :
- id, pwd, ls, hostname, ip a.
- vérifier que le démon dockerd fonctionne (ps -ef | grep -i dockerd).

Se déconnecter par 'exit'.

Inspecter les informations détaillées de la machine 'perso'.

Afficher l'adresse IP de la machine 'perso', puis son url et son status.

Identifier le nom de la machine active.
Prendre l'environnement de la machine 'perso'.
Identifier le nom de la machine active.
Exécuter un conteneur nommé alp1 en mode démon d'une image alpine.
Lister les images, puis les conteneurs en cours de fonctionnement.

Prendre l'environnement de la machine 'default'.
Identifier le nom de la machine active.
Lister les images, puis les conteneurs en cours de fonctionnement.
Prendre l'environnement de la machine 'perso'.
Supprimer le conteneur alp1 et l'image alpine.

Prendre l'environnement de la machine 'default'.
Redémarrer la machine 'perso'.
Vérifier qu'elle a bien redémarrée.

Arrêter la machine 'perso'. Vérifier.

Supprimer la machine 'perso'.

Exercice 3 : Déploiement customisé

Créer une machine virtuelle 'Oracle VirtualBox' via la commande docker-machine avec les caractéristiques suivantes : le nom est mavm,
1 Giga de RAM (1024 Mégas),
un disque de 2 Gigas (2048 Mégas).
Aidez-vous des aides de la commande pour identifier les options adéquates.

Utiliser les options de la commande docker-machine pour afficher des informations sur cette machine virtuelle.

Se connecter à la machine virtuelle et vérifier sa configuration : le hostname, la taille de la mémoire, la taille du disque et que le service de docker engine fonctionne. Puis se déconnecter.

Supprimer cette machine virtuelle.

Docker Swarm

Préalable :
Avoir les quatre machines virtuelles créées lors des exercices sur le thème de docker-machine :
master, worker1, worker2 et worker3.

Exercice 1 : Activation d'un cluster swarm

Afficher l'état de swarm par : docker info

Se connecter à la machine master.
Déclarer la machine master comme le leader du cluster.

Afficher l'état de swarm par : docker info

Lister les nœuds du cluster.

Se déconnecter du master.

Se connecter à la machine worker1.
Intégrer la machine worker1 au cluster.
Afficher l'état de swarm par : docker info
Se déconnecter de la machine worker1.

Refaire les mêmes manipulations pour intégrer au cluster : worker2 et worker3.

Prendre l'environnement du master par :

```
$ eval "$(docker-machine env master)"
```

Afficher l'état de swarm par : docker info

Lister les nœuds du cluster.

Exercice 2 : Management de services swarm

Lister les réseaux. Que remarquez-vous ?

Inspecter le réseau ingress.

Démarrer un service nginx nommé web dans le cluster (utiliser '-p 80:80' pour mapper les ports).

Lister les services (ls).
Lister les instances du service web (ps).
Tester le site via curl http://adresse_ip_master
Tester le site via curl avec les adresses des workers.
Consulter les logs du service web.
Inspecter le service web (avec et sans l'option –pretty).

Supprimer le service web.

Démarrer 8 services nginx nommé web dans le cluster (utiliser '-p 80:80' pour mapper les ports).
Utiliser l'option '--replicas N' de 'docker service' pou lancer N instances.

Lister les services.
Lister les instances du service web.
Tester le site via curl http://adresse_ip_master_ou_worker
Consulter les logs du service web.
Inspecter le service web (avec l'option –pretty).

Monter à 12 instances du service web.
Lister les services. Lister les instances du service web.

Descendre à 5 instances du service web.
Lister les services. Lister les instances du service web.

Revenir à 8 instances du service web.
Lister les services. Lister les instances du service web.

Vérifier le nombre d'instance qui fonctionne sur worker3.
Arrêter la machine worker3 (docker-machine stop worker3).
Lister les services. Lister les instances du service web.

Redémarrer la machine worker3.
Lister les services. Lister les instances du service web.

Supprimer tous les services web.

Exercice 3 : Management des masters swarm

Lister les nœuds du cluster.

Promouvoir les worker2 et worker3 en master.

Lister les nœuds du cluster, puis afficher les informations du cluster.

Démarrer 8 services nginx nommé web dans le cluster (utiliser '-p 80:80' pour mapper les ports).
Utiliser l'option '--replicas N' de 'docker service' pou lancer N instances.

Lister les services.
Lister les instances du service web.
Inspecter le service web (avec l'option –pretty).

Arrêter le master. Vérifier avec docker-machine ls.

Prendre l'environnement de worker3 (ou se connecter worker3).
Lister les nœuds du cluster, puis afficher les informations du cluster.
Lister les services. Lister les instances du service web.

Redémarrer le master. Vérifier.
Lister les instances du service web.

Monter à 12 instances du service web.
Lister les instances du service web.

Supprimer le service web.

Exercice 4 : Service répliqué et service global

--mode replicated : pour un service répliqué (valeur par défaut).
Le nombre de services est assuré même en cas de défaillance d'une machine.
--mode global : pour un service global.
Un service par machine du cluster.

Arrêter la machine worker1.
Lister les instances du service web.

Démarrer un service nommé web de l'image nginx en mode global (en mappant les ports).

Lister les services. Lister les instances du service web.

Démarrer la machine worker1.
Lister les services. Lister les instances du service web.

Démarrer 4 instances d'un service nommé alp1 de l'image alpine en mode répliqué qui exécute la commande 'ping 8.8.8.8'.

Lister les services. Lister les instances du service web et de alp1.

Arrêter la machine worker1.
Lister les services. Lister les instances du service web et de alp1.

Démarrer la machine worker1.
Lister les services. Lister les instances du service web.

Supprimer les service web et alp1.

CORRECTIONS DES EXERCICES

Correction - Installation et configuration de docker

Exercice 1 : L'installation de Docker

Installer le dépôt docker.

```
# yum-config-manager  --add-repo \
     https://download.docker.com/linux/centos/docker-ce.repo
```

Mette à jour les méta-données.

```
# yum  makecache  fast
```

Installer Docker.

```
# yum  install -y docker-ce
```

Démarrer Docker et l'activer pour le prochain redémarrage.

```
# systemctl start docker
# systemctl enable docker
```

Afficher le statut du service Docker.

```
# systemctl status docker
```

Afficher la version de Docker de manière succincte et détaillée.

```
# docker -v       ou       docker --version
# docker version
```

Vérifier qu'un groupe appelé docker a été créé.

```
# grep docker /etc/group
```

Vérifier qu'une interface réseau docker0 a été créé.

```
# ip a
```

Exercice 2 : L'aide de Docker

Afficher l'aide de la commande docker de deux manières différentes.

```
# docker  --help                  man  docker
```

Afficher l'aide de la sous-commande 'search' de deux manières différentes.

```
# docker  search  --help          man docker-search
```

Afficher l'aide de la sous-commande 'image' de deux manières différentes.

```
# docker  image  --help           man docker-image
```

Afficher l'aide de la sous-commande 'run' de deux manières différentes.

```
# docker  run  --help      man docker-run
```

Afficher l'aide de la sous-commande 'volume' de deux manières différentes.

```
# docker  volume  --help        man docker-volume
```

Afficher l'aide de 'create' de la sous-commande 'volume' de deux manières différentes.

```
# docker  volume  create  --help      man docker-volume-create
```

Correction – Docker en ligne de commandes

Préalable : exécuter les commandes suivantes :

```
# docker rm -f $(docker ps -qa)
# docker rmi $(docker images -q)
```

Exercice 1 : Les commandes de recherche

Rechercher les images ubuntu.

```
# docker search ubuntu
```

Rechercher les images ubuntu qui ont plus de 50 étoiles.

```
# docker search ubuntu --filter stars=50
```

Rechercher les images ubuntu en n'affichant que 5 résultats.

```
# docker search ubuntu --limit 5
```

Du résultat précédent, afficher en entier le champ de description.

```
# docker search ubuntu --limit 5 --no-trunc
```

Rechercher les images ubuntu qui sont officielles.

```
# docker search --filter "is-official=true" ubuntu
```

Exercice 2 : Exécution d'un conteneur

Exécuter un conteneur ubuntu en affichant le fichier /etc/passwd.

```
# docker run ubuntu cat /etc/passwd
```

Que s'est il passé ?
L'image a été téléchargé depuis le docker hub. Par défaut l'image nommée latest a été téléchargé.

Recommencer en affichant le fichier /etc/hosts.

```
# docker run ubuntu cat /etc/hosts
```

Quelle est la différence avec l'étape numéro 1 ?
L'image étant présente dans le cache local, elle n'a pas été téléchargé.

Combien de conteneurs sont présents ?

```
# docker ps -a
```

2 conteneurs sont présents. Ils ont été créé à partir de la même image.

Combien d'images sont présentes dans le cache local ?

```
# docker image ls
# docker images
```

Une seule image est présente, elle a servie pour créer les deux conteneurs.

Exécuter un conteneur nommé dock1 d'une image debian en mode démon (arrière plan) et interactif.

```
# docker run -di --name dock1 debian
```

Lister les conteneurs, ainsi que ceux en cours de fonctionnement.

```
# docker ps -a
# docker ps
```

Arrêter les conteneurs et les supprimer via la commande suivante :

```
# docker stop  nom_conteneur_1  nom_conteneur_2  nom_conteneur_3 . . .
# docker rm    nom_conteneur_1  nom_conteneur_2  nom_conteneur_3 . . .
```

Create / pull / start / exec

Lister les images nginx pour vérifier qu'il n'y en a pas de présente.

```
# docker images nginx
```

Transférer l'image nginx (la dernière version) dans le cache local de votre poste. Vérifier.

```
# docker pull nginx:latest
# docker images nginx
```

Créer un conteneur de l'image nginx sans le démarrer et nommé web1.
Lister les conteneurs (vérifier le status de web1).

```
# docker create --name web1 nginx
# docker ps -a
```

Démarrer le conteneur web1.
Lister les conteneurs, ainsi que ceux en cours de fonctionnement.

```
# docker start web1
# docker ps -a
# docker ps
```

Se connecter au conteneur web1 via la sous commande 'exec', les options -it et la commande /bin/bash.
Une fois connecté, taper les commandes hostname et 'ls /usr/share/nginx/html'.
Quitter le conteneur sans l'arrêter. Vérifier qu'il soit toujours en cours de fonctionnement.

```
# docker exec -it web1 /bin/bash
xxxx # hostname     puis    ls /usr/share/nginx/html
xxxx # crtl + PQ
# docker ps
```

Arrêter le conteneur web1 et le supprimer.

run / option dit et it

Exécuter un conteneur nommé cont1 d'une image debian en mode démon (arrière plan), interactif et en associant un terminal : -dit
Lister les conteneurs en cours de fonctionnement.
Exécuter un conteneur nommé cont2 d'une image debian en interactif et en associant un terminal : -it
Que constatez-vous ?
Reprendre la main avec 'CTRL + PQ'.
Lister les conteneurs en cours de fonctionnement. Le conteneur cont2 est-il arrêté ?

```
# docker run -dit --name cont1 debian
# docker ps
# docker run -it --name cont2 debian
 . . . ctrl+PQ
# docker ps
```

Utilise la sous commande attach pour se reconnecter au conteneur cont2.
Quitter avec 'CTRL + PQ'. Lister les conteneurs en cours de fonctionnement. Que constatez-vous ?
Utilise la sous commande attach pour se reconnecter au conteneur cont2.
Quitter avec 'exit'. Lister les conteneurs en cours de fonctionnement. Que constatez-vous ?

```
# docker attach cont2
 . . . ctrl+PQ
# docker ps
# docker attach cont2
xxxx # exit
# docker ps
```

Le conteneur est arrêté.

Supprimer le conteneur cont1, alors qu'il en cours de fonctionnement (sans utiliser la sous commande stop).

```
# docker rm -f cont1
```

Supprimer le conteneur cont2.

```
# docker rm  cont2
```

Exécuter la commande suivante : docker run -it --name cont1 debian ping 8.8.8.8
Faire 'CTRL + C'
Exécuter la commande suivante : docker run -dit --name cont2 debian ping 8.8.8.8
Lister les conteneurs.
Se connecter au conteneur cont2 par : docker attach cont2
Quitter avec 'CTRL + PQ'. Lister les conteneurs en cours de fonctionnement. Que constatez-vous ?
Utilise la sous commande attach pour se reconnecter au conteneur cont2.
Quitter avec 'CTRL + C'. Lister les conteneurs en cours de fonctionnement. Que constatez-vous ?

Faire un « grand nettoyage » via : docker system prune

Inspect / logs

Lister le contenu du répertoire /var/lib/docker/containers. Le répertoire est vide.

Exécuter un conteneur nommé monweb d'une image httpd avec les options (-dit)

```
# docker run -dit --name monweb httpd
```

Noter le début du nom qui s'est affiché comme résultat de la commande.

Lister les conteneurs, et noter le 'container id'.

Inspecter le conteneur.
Remarquer 'ID', 'State'.

Lister le contenu du répertoire /var/lib/docker/containers.

Exécuter la commande suivante : docker inspect -f '{{.Config.Hostname}}' monweb

Consulter la configuration réseau du conteneur monweb via 'docker inspect'.
Noter l'adresse IP et la gateway.
Vérifier son bon fonctionnement par : curl http://adresse_ip_conteneur_monweb

Renommer le conteneur monweb en monsite.
Vérifier.

```
# docker rename monweb monsite
```

Générer de l'activité par : curl http://adresse_ip_conteneur_monsite
Afficher les logs de monsite.

```
# docker logs monsite
```

Générer à nouveau de l'activité par :curl http://adresse_ip_conteneur_monsite
Afficher les logs de monsite.

Affichez les statistiques d'utilisation des ressources de monsite.

```
# docker stats monsite
```

Arrêter monsite.

```
# docker stop monsite
```

Exécuter un conteneur avec les caractéristiques suivantes :
nom : perso
de l'image debian
en mode démon, interactif et en affectant un speudo terminal
le conteneur se supprimera à son arrêt.
Il doit exécuter la commande : ping 8.8.8.8
Lister les conteneurs en cours de fonctionnement.
Utiliser les sous commandes top, puis stats, puis logs pour le conteneur 'perso'.

```
# docker run -dit --rm --name perso debian ping 8.8.8.8
```

```
# docker ps
# docker top perso
# docker stats perso
# docker logs perso
```

Afficher la différence entre le conteneur et son image. A ce stade, il ne doit rien afficher.

Se connecter au conteneur perso via : docker exec -it perso /bin/bash
Créer un fichier fic et supprimer /opt (touch /fic ; rm -rf /opt).
Quitter par CRL+PQ.
Afficher la différence entre le conteneur et son image.

```
# docker diff perso
```

En une commande, supprimer tous les conteneurs sans préciser le nom des conteneurs.

```
# docker rm -f $(docker ps -qa)
```

Exercice 3 : Les variables

Rechercher les images mariadb.

```
# docker search mariadb
```

Télécharger l'image de mariadb.

```
# docker pull mariadb
```

Exécuter en arrière plan un conteneur nommé sql0 de l'image mariadb.
Constater que le conteneur a été créé mais ne fonctionne pas. Démarrer le conteneur et constater qu'il est arrêté.

```
# docker run --name sql0 -d mariadb
# docker ps
# docker ps -a
# docker start sql0
# docker ps ;  docker ps -a
```

Exécuter en arrière plan un conteneur nommé sql1 de l'image mariadb avec les variables suivantes :
MYSQL_USER=user1
MYSQL_PASSWORD=mypass75
MYSQL_DATABASE=db1
MYSQL_ROOT_PASSWORD=mypass75

```
# docker run --name sql1 -d -e MYSQL_USER=user1 \
              -e MYSQL_PASSWORD=mypass75 -e MYSQL_DATABASE=db1 \
              -e MYSQL_ROOT_PASSWORD=mypass75   mariadb
```

Vérifier que le conteneur s'exécute bien.

```
# docker ps
```

Afficher l'adresse IP du conteneur.
```
# docker inspect -f '{{ .NetworkSettings.IPAddress }}' sql1
```

Installer la commande mysql.
```
# yum install -y mariadb
```

Connectez-vous à la base de données, puis exécuter les commandes suivantes : help, status, quit.
```
# mysql -u user1 -h adresse_ip -p db1
     help              status           quit
```

Créer un nouveau container appelé sql2 en regroupant les variables dans un fichier.
Connectez-vous à la base de données, puis exécuter les commandes suivantes : help, status, quit.
```
# cat var_mariadb
MYSQL_USER=user1
MYSQL_PASSWORD=mypass75
MYSQL_DATABASE=db1
MYSQL_ROOT_PASSWORD=mypass75
# docker run --name sql2 -d --env-file=var_mariadb mariadb
# docker inspect -f '{{ .NetworkSettings.IPAddress }}' sql2
# mysql -u user1 -h adresse_ip -p db1
     help              status           quit
```

Supprimer les trois conteneurs.
```
# docker rm -f sql0 sql1 sql2
```

Exercice pour le hostname

Démarrer un conteneur en arrière plan, nommé site1, de l'image nginx.
```
# docker run -d  --name site1 nginx
```

Démarrer un conteneur en arrière plan, nommé site2, de l'image nginx avec le hostname à client2.
```
# docker run -d  -h client2 --name site2 nginx
```

Vérifier leurs présences en cours d'exécution.
```
# docker ps
CONTAINER ID  IMAGE  COMMAND             CREATED             STATUS             PORTS
NAMES
9e355e80c33a  nginx  "nginx -g 'daemon…" 46 seconds ago      Up 45 seconds      80/tcp
site2
703341f709cc  nginx  "nginx -g 'daemon…" About a minute ago  Up About a minute  80/tcp
site1
```

Afficher les adresses IP des deux conteneurs.
```
# docker inspect -f '{{ .NetworkSettings.IPAddress }}' site1 site2
172.17.0.2
172.17.0.3
```

Afficher le hostname des deux conteneurs.
```
# docker inspect -f '{{ .Config.Hostname }}' site1 site2
703341f709cc
client2
```

Supprimer les deux conteneurs.
```
# docker rm -f site1 site2
```

Exercice 4 : Les copies, pause, unpause et wait

Préalable : le fichier index_base.html est fourni et il est localisé dans votre répertoire 'sources'.

Démarrer un conteneur en arrière plan, nommé site1, de l'image nginx.
Démarrer un conteneur en arrière plan, nommé site2, de l'image nginx.

```
# docker run -d  --name site1 nginx
# docker run -d  --name site2 nginx
```

Vérifier le bon fonctionnement en consultant la page de garde des deux conteneurs.
curl http://adresse_ip_du_conteneur
ou via votre browser http://adresse_ip_du_conteneur

```
# curl http://172.17.0.2
# curl http://172.17.0.3
```

Copier le fichier index_base.html de votre répertoire 'sources' au sein du conteneur site1 (le répertoire est /usr/share/nginx/html et le fichier est nommé index.html). Consulter la page de site.

```
# docker cp sources/index_base.html \
            site1:/usr/share/nginx/html/index.html
# curl http://172.17.0.2<html>
```

Essayer de copier le fichier index.html de votre conteneur site1 au sein du même répertoire du conteneur site2.

```
# docker cp site1:/usr/share/nginx/html/index.html \
            site2:/usr/share/nginx/html/index.html
copying between containers is not supported
```

Mettre en pause le conteneur site1. Consulter l'état par 'docker ps'. Consulter la page de site.

```
# docker pause site1
# docker ps
# curl http://172.17.0.2
```

Retirer le conteneur site1 du mode pause. Consulter l'état par 'docker ps' et la page de site.

```
# docker unpause site1
# docker ps
# curl http://172.17.0.2
```

Arrêter le conteneur site2.

```
# docker stop site2
```

Ouvrir une nouvelle fenêtre terminale et taper :

```
# echo "debut de mon script" ; docker wait site1 ; \
        echo "suite de mon script" ; docker start site2
```

Que fait cette séquence de commandes ?

De la première fenêtre terminale, vérifier l'état des conteneurs par 'docker ps -a'. Puis mettre en pause le conteneur site1. Que constatez-vous dans l'autre fenêtre terminale ?
vérifier l'état des conteneurs par 'docker ps -a'.

Retirer le conteneur site1 du mode pause. Que constatez-vous dans l'autre fenêtre terminale ?
vérifier l'état des conteneurs par 'docker ps -a'.
Arrêter le conteneur site1. Que constatez-vous dans l'autre fenêtre terminale ?
vérifier l'état des conteneurs par 'docker ps -a'.

```
# docker ps -a
     site1 fonctionne et site2 est arrêté.
# docker pause site1
     La commande est toujours en attente au sein de l'autre fenêtre.
# docker ps -a
# docker unpause site1
     La commande est toujours en attente au sein de l'autre fenêtre.
# docker ps -a
# docker stop site1
     La commande a continué son exécution au sein de l'autre fenêtre.
# docker ps -a
     site1 est arrêté et site2 a démarré.
```

Supprimer les deux conteneurs.

```
# docker rm -f site1 site2
```

Exercice 5 : Arrêt et démarrage d'un conteneur

Créer quatre conteneurs web d'après l'image de nginx appelés respectivement web_no, web_always, web_unless_stropped et web_on_failure. Utiliser pour chacun des conteneurs la valeur adéquate de l'option restart en fonction de leur nom.
Faites les fonctionner en mode démon.

```
# docker run -d --name web_no --restart no nginx
# docker run -d --name web_always --restart always nginx
# docker run -d --name web_unless_stopped --restart unless-stopped nginx
# docker run -d --name web_on_failure --restart on-failure nginx
```

Vérifier que les quatre containers s'exécutent.

```
# docker ps
```

Exécuter la commande 'pkill -9 nginx'.
Quels conteneurs sont redémarrés et ceux qui sont arrêtés ?

```
# pkill -9 nginx
# docker ps -a
```

Les conteneurs redémarrés sont : web_always, web_unless_stropped et web_on_failure.
Le conteneur web_no est arrêté.

Redémarrer le(s) conteneur(s) arrêté(s).

```
# docker start web_no
```

Redémarrer le service docker. Vérifier quels conteneurs ont redémarré et lesquels ont été arrêté.
Quels conteneurs sont redémarrés et ceux qui sont arrêtés ?

```
# systemctl restart docker
# docker ps -a
```

Les conteneurs web_no et web_on_failure sont arrêtés. Les conteneurs web_always et

web_unless_stopped sont démarrés.

Arrêter les conteneurs qui sont démarrés.
```
# docker stop web_always web_unless_stopped
```

Redémarrer le service docker (équivalent à un redémarrage de votre serveur).
Quels conteneurs sont redémarrés et ceux qui sont arrêtés ?
```
# systemctl restart docker
# docker ps -a
```
Tous les conteneurs sont arrêtés sauf web_always.

Supprimer les conteneurs de cet exercice.
```
# docker rm -f $(docker ps -qa)
```

Exercice 6 : Les volumes

Préalable : dans votre répertoire 'sources', les fichiers base.connec et annuaire.sql sont fournis.

Faire un « grand nettoyage » via :
```
# docker  system prune --volumes
```
Vérifier que le répertoire /var/local est vide.
```
# ls /var/local
```

Exécuter un conteneur en mode démon nommé bdd1 de l'image mariadb avec un volume persistant. Pour ce volume, les données de mariadb (répertoire /var/lib/mysql) devront être accessibles depuis le répertoire /var/local/mariadb de l'hôte.
Vous utiliserez également les variables du fichier base.connec.
```
# docker run --name bdd1 -d -v /var/local/mariadb:/var/lib/mysql \
             --env-file ~/Sources/base.connec  mariadb
```

Vérifiez que le container fonctionne et consulter le répertoire /var/local.
```
# docker ps
# ls /var/local
```
Un répertoire mariadb est présent avec les données dans /var/lib/mysql du conteneur.

Exécuter la commande suivante pour créer une table 'annuaire' dans la base de données :
```
# mysql -u user1 -h 172.17.0.2 -p db1 < ~/Sources/annuaire.sql
   saisir le mot de passe
```
Vérification par :
```
# mysql -u user1 -h 172.17.0.2 -p db1
   saisir le mot de passe
   select * from annuaire ;
   quit
```

Connectez-vous au conteneur (via la commande docker exec).
```
# docker exec -it bdd1 /bin/bash
```

Au sein du conteneur, créer un fichier quelconque (exemple : TEST.DB) dans /var/lib/mysql.
```
root@5028d12f03ae:/# touch /var/lib/mysql/TEST.DB
```

Déconnectez-vous du conteneur.
De votre machine hôte, le fichier apparaît-il dans /var/local/mariadb ?

```
root@5028d12f03ae:/# exit
# ls /var/local/mariadb
```

Le fichier apparaît bien.

Arrêter et supprimer le conteneur.

```
# docker stop bdd1
# docker rm bdd1
```

Essayer d'accéder à la base de donnée avec la commande suivante, vous aurez un échec.

```
# mysql -u user1 -h 172.17.0.2 -p db1
   saisir le mot de passe
```

Le répertoire /var/local/mariadb est-il présent ? Si oui, le fichier créé existe-t-il toujours ?
Oui.

Exécuter un nouveau conteneur en mode démon nommé autrebdd de l'image mariadb avec le même partage de volumes et avec le fichier base.connec.

```
# docker run --name autrebdd -d -v /var/local/mariadb:/var/lib/mysql \
             --env-file ~/Sources/base.connec mariadb
```

Connectez-vous au conteneur (via la commande docker exec).
Au sein du conteneur, le fichier créé précédemment dans /var/lib/mysql est-il présent ?
Quitter le conteneur.

```
# docker exec -it autrebdd /bin/bash
root@e445aa8b12aa:/# ls /var/lib/mysql
root@e445aa8b12aa:/# exit
```

Le fichier est présent.

Exécuter :

```
# mysql -u user1 -h 172.17.0.2 -p db1
   saisir le mot de passe
   select * from annuaire ;
   quit
```

Qu'en déduisez-vous ?
Les données sont bien persistantes après la destruction du container.

Lister les volumes par 'docker volume ls'.

```
# docker volume ls
```

Rien n'apparaît.

Afficher les informations détaillées de votre conteneur pour retrouver les informations du volume.

```
# docker inspect autrebdd
# docker inspect -f '{{.HostConfig.Binds}}'  autrebdd
[/var/local/mariadb:/var/lib/mysql]
# docker inspect -f '{{.Mounts}}'  autrebdd
[{bind  /var/local/mariadb /var/lib/mysql   true rprivate}]
```

Supprimer les conteneurs de cet exercice.

```
# docker rm -f $(docker ps -qa)
```

Exercice 7 : Les ports réseaux

Préalable : les fichiers index_base.html et index_autre.html sont dans votre répertoire 'sources'.

Dans votre répertoire de connexion (/root), créer deux répertoires rep_a et rep_b.
Copier dans les deux répertoires le fichier index_base.html en le renommant index.html.

```
# cd ; mkdir rep_a rep_b
# cp Sources/index_base.html rep_a/index.html
# cp Sources/index_base.html rep_b/index.html
```

Exécuter 4 conteneurs de l'image nginx :
- site2 : en mappant le port 92 du poste local au port 80 du conteneur,
  en mappant le répertoire rep_a du poste local avec /usr/share/nginx/html du conteneur.
- site3 : en mappant le port 93 du poste local au port 80 du conteneur,
  en mappant le répertoire rep_a du poste local avec /usr/share/nginx/html du conteneur.
- site4 : en mappant le port 94 du poste local au port 80 du conteneur,
  en mappant le port 194 du poste local au port 443 du conteneur,
  en mappant le répertoire rep_b du poste local avec /usr/share/nginx/html du conteneur.
- site5 : en mappant le port 95 du poste local au port 80 du conteneur,
  en mappant le port 195 du poste local au port 443 du conteneur,
  en mappant le répertoire rep_b du poste local avec /usr/share/nginx/html du conteneur.

```
# docker run -d --name site2 -v /root/rep_a:/usr/share/nginx/html \
                -p 92:80 nginx
# docker run -d --name site3 -v /root/rep_a:/usr/share/nginx/html \
                -p 93:80 nginx
# docker run -d --name site4 -v /root/rep_b:/usr/share/nginx/html \
                -p 94:80 -p 194:443 nginx
# docker run -d --name site5 -v /root/rep_b:/usr/share/nginx/html \
                -p 95:80 -p 195:443 nginx
```

Lister les conteneurs en cours de fonctionnement, consulter la section PORTS.
Lister les ports du conteneur site5

```
# docker ps
# docker port site5
```

Inspecter en détail le conteneur site5 pour retrouver les informations du mappage des ports.

```
# docker inspect site5
# docker inspect -f '{{.HostConfig.PortBindings}}' site5
map[443/tcp:[{ 195}] 80/tcp:[{ 95}]]
# docker inspect -f '{{.NetworkSettings.Ports}}' site5
map[443/tcp:[{0.0.0.0 195}] 80/tcp:[{0.0.0.0 95}]]
```

Afficher la page de garde de chaque conteneur.

```
# curl http://localhost:92            curl http://localhost:93
# curl http://localhost:94            curl http://localhost:95
```

Modifier le fichier index.html du répertoire rep_b en récupérant le fichier index_autre.html.

```
# cp Sources/index_autre.html rep_b/index.html
```

Afficher la page de garde de chaque conteneur.

```
# curl http://localhost:92            curl http://localhost:93
# curl http://localhost:94            curl http://localhost:95
```

Redémarrer le conteneur site5. Lister les ports.

```
# docker restart site5
# docker port site5
```

Exécuter un conteneur site6 de l'image nginx en mappant les ports de nginx aléatoirement avec les ports du poste hôte.
Afficher les ports de site6.

```
# docker run -d --name site6 -P nginx
# docker port site6
```

Supprimer les conteneurs de cet exercice.

```
# docker rm -f $(docker ps -qa)
```

Exercice 8 : Gestion des ressources

Exercice 7.1 : Gestion des ressources CPU

Créer trois conteneurs basés sur une Débian appelés respectivement deb1, deb2 et deb3 avec les contraintes suivantes :

- Affecter 10 parts de CPU à deb1
- Affecter 30 parts de CPU à deb2
- Affecter 60 parts de CPU à deb3

Connectez-vous aux conteneurs. Exécutez la commande 'yes > /dev/null &' dans chacun des conteneurs. Cette commande va générer une saturation de l'utilisation CPU.

```
# docker run --name deb1 -it --cpu-shares=10 debian /bin/bash
root@5576772d16a7:/# yes > /dev/null &
# docker run --name deb2 -it --cpu-shares=30 debian /bin/bash
root@ee86bf1609ea:/# yes > /dev/null &
# docker run --name deb3 -it --cpu-shares=60 debian /bin/bash
root@cd79cb28108b:/# yes > /dev/null &
```

Sur la machine hôte afficher le taux d'utilisation CPU (commande top).
Vérifier que deb1, deb2 et deb3 utilisent respectivement 10% , 30% et 60% des ressources CPU.

```
# top
```

On constate que les pourcentages sont corrects.

Arrêter le conteneur deb3. Sur la machine hôte afficher le taux d'utilisation CPU, vérifier que la répartition de charge respecte les parts.

Supprimer les 3 conteneurs.

Exercice 7.2 : Gestion des ressources mémoire

Exécuter la commande 'free -h' pour constater la mémoire totale du système hôte.

Démarrer un conteneur nommé cent1 d'une image centos avec une limitation mémoire à 100 mégas.
Au sein de ce conteneur :
- installer les packages epel-release, puis stress.
- exécuter la commande 'free -h' pour vérifier l'utilisation mémoire et la mémoire totale.
- générer une utilisation mémoire de 100 mégas par la commande :
  stress --vm 1 --vm-bytes 100M &
- vérifier l'utilisation mémoire de la commande précédente :
  ps -aux la colonne %MEM par rapport à la mémoire totale
- arrêter la commande stress : pkill stress
- générer une utilisation mémoire de 200 mégas par la commande :
  stress --vm 1 --vm-bytes 200M &
- vérifier l'utilisation mémoire de la commande précédente :
  ps -aux la colonne %MEM par rapport à la mémoire totale
  Que constatez-vous ?
- arrêter la commande stress : pkill stress

Manipulations avec l'option -m :

```
# free -h
              total        used        free      shared  buff/cache   available
Mem:           1.8G        711M        411M        9.9M        716M        902M
Swap:          2.0G         26M        2.0G

# docker  run  -it  -m 100M  --name cent1  centos
[root@7f9c5b34e336 /]# yum install -y epel-release
[root@b018c1c0dd27 /]# yum install -y stress
[root@b018c1c0dd27 /]# free -h
              total        used        free      shared  buff/cache   available
Mem:           1.8G        711M        411M        9.9M        716M        902M
Swap:          2.0G         26M        2.0G

[root@7f9c5b34e336 /]# stress --vm 1  --vm-bytes 100M &
[root@7f9c5b34e336 /]# ps -aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       152 22.1  4.5 109668 86184 pts/0    D    12:41   0:01 stress --vm 1 -
      Ce qui correspond : 100Mb / ( 1.8G * 1024 ) = 0.05 , soit 5%

[root@7f9c5b34e336 /]# pkill stress
[root@7f9c5b34e336 /]# stress --vm 1  --vm-bytes 200M &
[root@7f9c5b34e336 /]# ps -aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       174 23.8  4.6 160868 87512 pts/0    D    12:46   0:01 stress --vm 1 -
      Impact de l'option '-m 100M' , sinon on aurait du avoir approximativement 10%

[root@7f9c5b34e336 /]# pkill stress
[root@7f9c5b34e336 /]# exit
```

Supprimer le conteneur.

```
# docker rm -f cent1
```

Correction – Création d'une image personnalisée

Préalable :
Arrêter tous les conteneurs et les supprimer, ainsi que les volumes.

```
# docker stop $(docker ps -q)
# docker system prune --volumes
```

Exercice 1 : Recherche de versions d'images

Afficher tous les tags pour l'image 'centos'.

```
# curl https://registry.hub.docker.com/v1/repositories/centos/tags
```

Afficher tous les tags de pour l'image 'mariadb'.

```
# curl https://registry.hub.docker.com/v1/repositories/mariadb/tags
```

Exercice 2 : Création d'une image à partir d'un container

Exécuter un container ubuntu appelé ubu1 et connectez-vous dedans.

```
# docker run --name ubu1 -it ubuntu
```

Créer les utilisateurs user1 et user2 avec leur répertoire de connexion.

```
root@b3c96210418d:/# useradd -m user1
root@b3c96210418d:/# useradd -m user2
```

Sortir du conteneur et créer une image appelée monimage à partir de celui-ci.

```
# docker commit ubu1 monimage
```

Lister l'image. Afficher les informations détaillées de cette image.

```
# docker images monimage
# docker image ls monimage
# docker image inspect monimage
        "Id": "sha256:9bd0d3910d5be7ee5fe7eaf12f1e59b742fa97af5be66b905c201528d4a1c301",
        "Parent": "sha256:452a96d81c30a1e426bc250428263ac9ca3f47c9bf086f876d11cb39cf57aeec",
        "Container": "0acdcaf4f129ae040441e69226a29d553cdc39c7c70372025db4303e514537ab",
        "Config": {
            "Hostname": "0acdcaf4f129",
...
```

Afficher l'historique de l'image monimage.

```
# docker image history monimage
# docker history  monimage
IMAGE               CREATED             CREATED BY                                      SIZE
9bd0d3910d5b        3 minutes ago       /bin/bash                                       403kB
452a96d81c30        5 months ago        /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>           5 months ago        /bin/sh -c mkdir -p /run/systemd && echo 'do…   7B
. . .
```

Consulter les répertoires /var/lib/docker, /var/lib/docker/containers, /var/lib/docker/image, ...

Analyser les résultats de inspect (champs ID, Parent, Container, Hostname, …) avec ceux de history (colonne IMAGE) et de l'arborescence /var/lib/docker.

Créer un nouveau conteneur appelé ubu2 à partir de monimage.

```
# docker run --name ubu2 -it monimage
```

Les utilisateurs créés dans ubu1 sont-ils présents au sein de ubu2 ?

```
root@153f6b51da3e:/# grep user /etc/passwd
```

Les utilisateurs user1 et user2 existent bien.

Sortir du conteneur ubu2, lister les conteneurs. Pour ubu1 et ubu2, noter les container_id.
Consulter le contenu de /var/lib/docker/containers.

Créer le tag 1.0 pour l'image monimage. Lister les images monimage.
Créer une image monimage:2.0 à partir de ubu2. Au préalable arrêter ubu2. Lister les images.
Supprimer l'image monimage:latest. Lister les images monimage.
Créer le tag latest pour l'image 2.0. Lister les images monimage.

```
# docker tag monimage monimage:1.0
# docker images monimage
# docker stop ubu2
# docker commit ubu2 monimage:2.0
# docker images monimage
# docker rmi monimage:latest
# docker images monimage
# docker tag monimage:2.0 monimage:latest
# docker images monimage
```

Utiliser la sous commande history sur l'image 1.0 puis sur l'image 2.0. Que constatez-vous ?

```
# docker history monimage:1.0
# docker history monimage:2.0
```

Les couches de la version 2.0 sont constituées de celles de la version 1.0 plus une couche.

Lister le contenu de /var/lib/docker/containers.

Il y a les 2 répertoires des conteneurs.

Supprimer les deux conteneurs ubu1 et ubu2.
Lister le contenu de /var/lib/docker/containers.

Le répertoire est vide.

```
# ls /var/lib/docker/containers
# docker rm -f ubu1 ubu2
# ls /var/lib/docker/containers
```

Supprimer les images monimage.

```
# docker rmi monimage:latest monimage:1.0 monimage:2.0
```

Exercice 3 : Création d'une image personnalisée

Préalable : le fichier Dockerfile est fourni dans le répertoire de l'exercice.

Soit le fichier Dockerfile suivant.

```
# more Dockerfile
FROM centos:latest
RUN yum update -y
RUN yum install -y wget gcc make
RUN wget https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
RUN tar zxf hello-2.10.tar.gz
RUN cd hello-2.10  && ./configure && make && make install
RUN yum remove -y wget gcc make
```

Analyser le fichier Dockerfile.
Utilisation d'une image centos, installation de packages et de la commande hello. Enfin, suppression de packages inutiles au fonctionnement d'un conteneur.

Créer une image appelée testimg:1.0 à partir de celui-ci.

```
# docker build -t testimg:1.0 .
```

Afficher la taille de l'image.

```
# docker images testimg:1.0
```

L'image a une taille d'environ 600M.

Exécuter un conteneur pour vérifier que la commande hello fonctionne. Utiliser l'option supprimant le conteneur à son arrêt.

Syntaxe de la commande : hello [-g autre_message]

```
# docker run -it --rm testimg:1.0
[root@728ac2ef58fd /]# hello
[root@728ac2ef58fd /]# hello -g «bonne journee»
[root@728ac2ef58fd /]# exit
# docker run -it --rm testimg:1.0 hello
# docker run -it --rm testimg:1.0 hello -g "bonne journee"
# docker ps -a
```

Exercice 4 : QCM d'optimisation de l'image

Sélectionner les possibilités qui permettent d'optimiser l'image précédente :
- supprimer la ligne '`RUN yum install -y wget gcc make`' car les packages sont supprimées à la fin. On peut donc retirer également la ligne '`RUN yum remove -y wget gcc make`'.
- utiliser un seul RUN et séquencer les commandes avec des &&.
- utiliser 3 RUN. Le 1er pour le 'yum update' et le 'yum install'. Le 2eme pour les trois RUN suivants et le 3eme pour le dernier RUN.
- ne pas utiliser l'image centos mais 'FROM /bin/bash'.
- ne pas utiliser l'image centos mais plutôt l'image alpine.
- ajouter une ligne RUN en fin de fichier pour retirer également les mans.
- avoir un FROM centos (ou alpine) avec un volume pour pointer sur le répertoire du hôte où est localisée la commande hello.

Exercice 5 : Optimisation de l'image

Préalable : le fichier Dockerfile est fourni dans le répertoire de l'exercice.

```
# more Dockerfile
FROM alpine
RUN apk update
RUN apk add wget build-base
RUN wget https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
RUN tar zxf hello-2.10.tar.gz
RUN cd hello-2.10  && ./configure && make && make install
RUN apk del wget build-base
```

Le fichier Dockerfile est la même version que précédemment. Au lieu d'utiliser une image CentOS, il est choisi une image Alpine.

Créer une image appelée testimg:2.0 à partir de ce fichier Dockerfile.
Vérifier la taille de l'image créée, la comparer à la taille de l'image 1.0.

```
# docker build -t testimg:2.0 .
# docker images testimg
```

L'image a une taille d'environ 160M.

Affichez l'historique de création de l'image.

```
# docker image history testimg:2.0
# docker history testimg:2.0
```

Modifier le fichier Dockerfile pour créer l'image la plus petite possible appelée testimg:3.0

```
# more Dockerfile
FROM alpine:latest
RUN apk update \
    && apk add wget build-base \
    && wget https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz \
    && tar zxf hello-2.10.tar.gz \
    && cd hello-2.10  && ./configure && make && make install \
    && apk del wget build-base
```

Créer l'image testimg:3.0 à partir du nouveau fichier Dockerfile. Remarque : l'option -f de 'docker build' permet d'indiquer un fichier différent de Dockerfile pour la construction de l'image.
Vérifier la taille de l'image créée, la comparer aux tailles des images précédentes.

```
# docker build -t testimg:3.0 .
Ou # docker build -t testimg:3.0 -f Dockerfile_solution .
# docker images testimg
```

L'image a une taille d'environ 12M.

Afficher l'historique de création de l'image .

```
# docker history testimg:3.0
```

Qu'en déduisez vous ?

Supprimer les images testimg.

```
# docker rmi testimg:1.0 testimg:2.0 testimg:3.0
```

Exercice 6 : Création d'une image à partir de rien (from scratch)

Préalable : le binaire salut est fourni dans le répertoire de l'exercice.

Vérifier le fonctionnement du binaire salut fourni.

```
# ./salut
```

Créer une image appelé 'light' à partir de rien (from scratch).
Pour cela, le Dockerfile est constitué avec :

- le mot clé scratch à la place du nom de l'image
- l'ajout (ADD) du binaire à la racine
- l'exécution de la commande par /salut avec CMD

```
# more Dockerfile
FROM scratch
ADD salut /
CMD ["/salut"]

# docker build -t light .
```

Afficher la taille de l'exécutable salut.
Afficher la taille de l'image.

```
# ls -lh salut
# docker images light
```

Qu'en déduisez-vous ?
La taille de l'image dépend de la source et de la constitution du Dockerfile.

Tester cette image en exécutant un conteneur :

```
# docker run --rm light
```

Supprimer l'image light.

```
# docker rmi light
```

Exercice 7 : Utilisation du Dockerfile

Préalable : les fichiers index_base.html et index_autre.html sont fournis dans le répertoire de l'exercice.

Préparer un fichier Dockerfile pour créer une image avec les informations suivantes :

- L'image sera la version latest de nginx
- On souhaite intégrer les informations suivantes :
  - maintainer avec votre nom
  - societe avec le nom de votre entreprise
  - version avec la version de votre fichier (exemple : 1.0)
  - description pour une présentation (exemple : Ceci est un exemple)
- Deux variables qui devront être exploitables au sein d'un conteneur :
  - ville avec le nom d'une ville de votre choix
  - pays avec France

- Une variable qui doit être utilisable lors de la création de l'image mais pas dans un conteneur : planete avec une planète de votre choix
- Copier le fichier index_base.html dans l'image à /usr/share/nginx/html/index.html
- Créer un répertoire /opt/contenu_de_la_variable_planete
- Au sein du répertoire créé ci-dessus, créer un fichier vide nommé avec le contenu de la variable pays
- Exposer les ports 80 et 443
- Créer deux volumes pour /usr/share/nginx/html et /var/log/nginx
- le répertoire de travail doit être positionné à /usr/share/nginx

```
# cat Dockerfile
FROM nginx:latest

LABEL maintainer="Baranger Jean-Marc" \
      societe="Spherius" \
      version="1.0" \
      description="Ceci est un exemple."

ENV   ville="Paris" pays="France"
ARG   planete="Terre"

COPY  index_base.html /usr/share/nginx/html/index.html
RUN   mkdir /opt/${planete} \
      && touch /opt/${planete}/${pays}

EXPOSE 80 443
VOLUME /usr/share/nginx/html \
       /var/log/nginx

WORKDIR /usr/share/nginx
```

Créer l'image nommée site à partir de votre fichier Dockerfile.

```
# docker image build -t site .
```

Consulter les informations détaillées de l'image.

```
# docker inspect site
```

Lister le contenu du répertoire /var/lib/docker/volumes.
Exécuter un conteneur en mode démon, nommé web, de cette image.
Lister le contenu du répertoire /var/lib/docker/volumes .

```
# ls /var/lib/docker/volumes
# docker run -dit --name web site
# ls /var/lib/docker/volumes
```

Lister le contenu des deux répertoires /var/lib/docker/containers/*/_data

```
# ls -R /var/lib/docker/volumes/*/_data
```

Faire un 'docker ps'.
Consulter les informations détaillées du conteneur web.

```
# docker ps
# docker inspect web
```

Consulter la page du site.

```
# curl http://ip_du_conteneur
```

Se connecter au conteneur avec 'docker exec -it web /bin/bash', puis effectuer les contrôles suivants :

```
# pwd
# ls -Rl /opt
# echo $ville
# echo $pays
# echo $planete
```

Quitter le conteneur.

```
# exit
```

Supprimer le conteneur, les volumes et l'image.

```
# docker rm -f web
# docker system prune --volumes
# docker rmi site
```

Exercice 8 : Utilisation du cache

Préalable : le fichier est fourni au sein du répertoire de l'exercice.

Se positionner dans le répertoire de l'exercice.
Consulter le fichier Dockerfile.

Vérifier que vous n'avez pas d'image centos, si c'est le cas la supprimer.

```
# docker images centos
# docker rmi centos
```

Créer une image test en utilisant la commande suivante (time permet d'afficher le temps total d'exécution de la commande) :

```
# time docker image build -t test .
```

Noter le temps d'exécution.

Recréer l'image de la même manière et noter le temps d'exécution. Que remarquez-vous ?

```
# time docker image build -t test .
```

Recréer l'image de la même manière avec l'option --no-cache et noter le temps d'exécution.
Que remarquez-vous ?

```
# time docker image build -t test --no-cache .
```

Ne pas supprimer l'image 'test'.

Exercice 9 : Sauvegarde / Restauration d'une image

Vérifier la présence de l'image test.

```
# docker images test
```

Sauvegarder l'image test dans un fichier monimage.save.

```
# docker  save  -o  monimage.save  test
# ls -lh monimage.save
```

Supprimer l'image test.

```
# docker rmi test
# docker images test
```

Restaurer l'image test.

```
# docker  load  -i  monimage.save
```

Vérifier.

```
# docker images test
```

Supprimer le fichier de sauvegarde et l'image.

```
# rm -f monimage.save
# docker rmi test
```

Correction – Docker et le réseau

Exercice 1 : Manipulation de réseaux

Créer 2 réseaux de type bridge nommés reseau1 et reseau2.
Lister les réseaux Docker et inspecter les 2 réseaux créés.

```
# docker network create reseau1
# docker network create reseau2
# docker network ls
# docker network inspect reseau1
# docker inspect reseau2
```

Exécuter deux conteneurs de l'image alpine en mode démon, nommés alp1 et alp2, et connectés au réseau 'reseau1'.
Exécuter un conteneur de l'image alpine en mode démon nommé alp3, connecté au réseau 'reseau2' et son hostname sera alpine3.
Inspecter la configuration réseau de ces conteneurs, et noter leurs adresses IP.

```
# docker run -dit --name alp1 --network reseau1 alpine
# docker run -dit --name alp2 --network reseau1 alpine
# docker run -dit --name alp3 --network reseau2 -h alpine3 alpine

# docker inspect -f '{{.NetworkSettings.Networks.reseau1.IPAddress}}' alp1 alp2
# docker inspect -f '{{.NetworkSettings.Networks.reseau2.IPAddress}}' alp3
```

Se connecter au conteneur alp1 et analyser sa configuration réseau avec entre autres les commandes suivantes : hostname ; ip a ; ip route et cat /etc/hosts.
Pinguez-vous le conteneur alp2 et alp3, un site externe (8.8.8.8 et www.google.fr), le hôte ?
Se déconnecter du conteneur avec 'ctrl+PQ'.

```
# docker attach alp1
/ # ping adresse_ip_alp2 et ping alp2 fonctionnent
/ # ping adresse_ip_alp3 ne fonctionne pas
/ # hostname ; cat /etc/hosts ; ip a ; ip route
```

Consulter la configuration des cartes réseaux de votre hôte (ip a).

```
# ip a
```

Connecter le conteneur alp1 au réseau 'reseau2'.
Inspecter la configuration réseau du conteneur alp1.
Se connecter au conteneur alp1 et analyser sa configuration réseau avec entre autres les commandes suivantes : hostname ; ip a ; ip route, cat /etc/hosts et cat /etc/resolv.conf.
Que constatez-vous ?
Pinguez-vous le conteneur alp3 ?
Se déconnecter du conteneur avec 'ctrl+PQ'.

```
# docker network connect reseau2 alp1
# docker inspect alp1
# docker attach alp1
```

```
/ # ping adresse_ip_alp2 et ping alp2 fonctionnent
/ # ping adresse_ip_alp3 et ping alp3 fonctionnent
/ # hostname ; cat /etc/hosts ; ip a ; ip route
```

Inspecter les réseaux 'reseau1' et 'reseau2'. Que constatez-vous ?

```
# docker network inspect reseau1
# docker network inspect reseau2
```

Les conteneurs associés à chaque réseau sont listés.

Déconnecter le réseau 'reseau2' du conteneur alp1.
Se connecter au conteneur alp1 et exécuter la commande 'ip a'.
Pinguez-vous le conteneur alp3 ?
Se déconnecter du conteneur avec 'ctrl+PQ'.

```
# docker network disconnect reseau2 alp1
# docker attach alp1
/ # ping adresse_ip_alp3 ne fonctionne pas
```

Supprimer les conteneurs.

```
# docker rm -f alp1 alp2 alp3
```

Exercice 2 : Configuration DNS

Exécuter un conteneur de l'image alpine en mode démon nommé alp1, sans spécifier de connexion à un réseau et son hostname sera alpine1.
Exécuter un conteneur de l'image alpine en mode démon nommé alp2, connecté au réseau 'reseau1' et son hostname sera alpine2.
Inspecter la configuration réseau de ces conteneurs.

```
# docker run -dit --name alp1 -h alpine1 alpine
# docker run -dit --name alp2 --network reseau1 -h alpine2 alpine
# docker inspect alp1 alp2
```

Consulter le fichier /etc/resolv.conf du hôte et des deux conteneurs.

```
# cat /etc/resolv.conf
# docker exec alp1 cat /etc/resolv.conf
# docker exec alp2 cat /etc/resolv.conf
```

Le conteneur alp1 a le même fichier de celui du hôte.
Le conteneur alp2 étant connecté à un réseau spécifique (user defined) a un serveur DNS en 127.0.0..11.

Renommer le fichier /etc/resolv.conf du hôte en /etc/resolv.conf.old
Exécuter un conteneur de l'image alpine en mode démon nommé alp3, sans spécifier de connexion à un réseau.
Exécuter un conteneur de l'image alpine en mode démon nommé alp4, sans spécifier de connexion à un réseau et en spécifiant l'adresse ip 10.10.10.10 comme serveur dns (option '--dns').
Consulter le fichier /etc/resolv.conf des quatre conteneurs.

```
# mv /etc/resolv.conf /etc/resolv.conf.old
# docker run -dit --name alp3 alpine
# docker run -dit --name alp4 --dns 10.10.10.10 alpine
# docker exec alp1 cat /etc/resolv.conf
# docker exec alp2 cat /etc/resolv.conf
```

```
# docker exec alp3 cat /etc/resolv.conf
# docker exec alp4 cat /etc/resolv.conf
```

Renommer le fichier /etc/resolv.conf.old du hôte en /etc/resolv.conf

```
# mv /etc/resolv.conf.old /etc/resolv.conf
```

Supprimer les conteneurs.

```
# docker rm -f alp1 alp2 alp3 alp4
```

Exercice 3 : Configuration customisée d'un réseau

Créer un réseau avec les spécifications suivantes :

- nom : reseau3
- driver : bridge
- sous réseau 192.168.100.0
- netmask : 255.255.255.0 (/24)
- gateway : 192.168.100.254
- ce réseau devra pouvoir avoir 7 adresses IP de 1 à 7 (192.168.100.0/29)

Lister les réseaux et inspecter le réseau3.

```
# docker network create --driver bridge --subnet 192.168.100.0/24 \
       --gateway 192.168.100.254 --ip-range 192.168.100.0/29 reseau3
# docker network ls
# docker network inspect reseau3
```

Exécuter un conteneur de l'image nginx en mode démon nommé web1 et connecté au réseau 'reseau3'. Consulter sa configuration réseau et son adresse IP.

```
# docker run -dit --name web1 --network reseau3 nginx
# docker inspect web1
# docker inspect -f '{{.NetworkSettings.Networks.reseau3.IPAddress}}' web1
```

Exécuter six autres conteneurs de l'image nginx en mode démon nommés de web2 à web7 et connectés au réseau 'reseau3'.

```
# docker run -dit --name web2 --network reseau3 nginx
# docker run -dit --name web3 --network reseau3 nginx
# docker run -dit --name web4 --network reseau3 nginx
# docker run -dit --name web5 --network reseau3 nginx
# docker run -dit --name web6 --network reseau3 nginx
# docker run -dit --name web7 --network reseau3 nginx
```

Consulter les adresses IP de ces conteneurs.

```
# docker inspect -f '{{.NetworkSettings.Networks.reseau3.IPAddress}}' \
                  web1 web2 web3 web4 web5 web6 web7
```

Exécuter un autres conteneur de l'image nginx en mode démon nommés de web8 et connecté au réseau 'reseau3'.

```
# docker run -dit --name web8 --network reseau3
nginx31b9ef292ca1a07d30ca55b3005e409e3b7047d3ce917e003510557f4460a1ac
docker: Error response from daemon: no available IPv4 addresses on this
```

```
network's address pools: reseau3
(0611dff3454f0d8bef9b12a341a287f2c7a44e54fc2e3763387609526ec22516).
```

Que constatez-vous ?
Vérifier avec 'docker ps', 'docker ps -a', 'docker inspect web8' et 'docker inspect reseau3'.

```
# docker ps
# docker ps -a
# docker inspect web8
# docker inspect reseau3
```

Le conteneur web8 a été créé, il a bien été associé au réseau 'reseau3' mais il n'a pas démarré et ne possède pas d'adresse IP.

Exercice 4 : Suppression

Supprimer le réseau 'reseau3'.

```
# docker network rm reseau3
```

Que constatez-vous ?
Échec de la suppression car il est utilisé par au moins un conteneur.

Arrêter tous les conteneurs nginx.

```
# docker stop web1 web2 web3 web4 web5 web6 web7 web8
```

Vérifier qu'il n'y a plus de conteneurs associés au réseau 'reseau3'.

```
# docker inspect reseau3
```

Utiliser le mot clé 'prune' de la commande 'docker network' pour supprimer tous les réseaux qui ne sont pas utilisés.

```
# docker network prune
```

Supprimer les conteneurs alpine (nommés alpX) et supprimer les réseaux créés sur ce module ('reseau1', 'reseau2' et 'reseau3').

```
# docker rm -f $(docker ps -qa)
# docker network rm reseau1
Si nécessaire : # docker network rm reseau2
```

Correction – Docker et le stockage

Exercice 1 : Utilisation d'un volume

Préalable : le fichier /dev/log est un fichier spécial utilisé par le mécanisme syslog.

Objectif de l'exercice : on va associer le fichier spécial /dev/log du hôte à celui d'un conteneur. Ainsi en envoyant un message au mécanisme syslog du conteneur (via la commande logger), le syslog du hôte va les recevoir et les traiter en les plaçant dans ses fichiers de logs.

Créer un conteneur nommé cent1 basé sur une CentOS qui partage le fichier /dev/log avec le fichier /dev/log/ de la machine hôte. Le conteneur devra s'exécuter en mode interactif.

```
# docker run -it --name cent1 -v /dev/log:/dev/log centos
```

Depuis le conteneur avec la commande logger, envoyer un message à syslogd.

```
[root@94741b841502 /]# logger -p daemon.info "Test Log"
[root@94741b841502 /]# logger -p authpriv.info "Test Log securite"
```

Vérifier l'absence des fichiers /var/log/messages et /var/log/secure au sein du conteneur.

```
[root@94741b841502 /]# ls /var/log
```

Quitter le conteneur avec 'CTRL+PQ'.

Vérifier que sur la machine hôte vous avez accès aux logs du conteneur et consulter la dernière lignes de ces fichiers.

```
# tail -1 /var/log/messages
# tail -1 /var/log/secure
```

Inspecter le conteneur pour consulter la section 'Mounts'.

```
# docker inspect cent1
```

Supprimer le conteneur cent1.

```
# docker rm -f cent1
```

Exercice 2 : Création d'un conteneur et d'un volume en lecture seule

Exécuter un conteneur en mode interactif nommé cent1 d'une image centos en partageant le répertoire /data du hôte avec /data du conteneur.
A noter que le répertoire /data du hôte n'existe pas.

```
# ls /data      erreur car le répertoire n'existe pas.
# docker run -it --name cent1 -v /data:/data centos
```

Vérifier que le volume est monté.

```
[root@1c0b0bc57e06 /]# df -h
```

Créer un fichier vide fic1 dans / et /data.

```
[root@1c0b0bc57e06 /]# touch /fic1
[root@1c0b0bc57e06 /]# touch /data/fic1
```

Quittez le container avec 'exit'. Supprimer le conteneur cent1.
Vérifier que le fichier fic1 est présent sous /data du hôte.

```
[root@1c0b0bc57e06 /]# exit
# docker rm cent1
# ls /data
```

Exécuter un conteneur en mode interactif nommé cent1 d'une image centos en utilisant l'option --read-only et en partageant le répertoire /data du hôte avec /data du conteneur.
Créer un fichier vide fic2 dans / et /data. Que constatez-vous ?
Le répertoire /data est-il en lecture seule dans le container ?
Quittez le container avec 'exit'. Supprimer le conteneur cent1.
Vérifier que le fichier fic2 est présent sous /data du hôte.

```
# docker run --read-only -it --name cent1 -v /data:/data centos
[root@a061b37a2e8a /]# touch /fic2 /data/fic2
[root@a061b37a2e8a /]# exit
# docker rm cent1
# ls /data
```

C'est le container qui est en lecture seule. La création /fic2 a échoué.

Exécuter un conteneur en mode interactif nommé cent1 d'une image centos en partageant le répertoire /data du hôte avec /data du conteneur en lecture seule (l'option :ro au nom du volume du conteneur).
Créer un fichier vide fic3 dans / et /data. Que constatez-vous ?
Le répertoire /data est-il en lecture seule dans le container ?
Quittez le container avec 'exit'. Supprimer le conteneur cent1.
Vérifier le contenu de /data du hôte.

```
# docker run -it --name cent1 -v /data:/data:ro centos
[root@9741fe6a5ee6 /]# touch /fic3 /data/fic3
[root@9741fe6a5ee6 /]# exit
# docker rm cent1
# ls /data
```

Seul le répertoire /data est en lecture seule.
La création de /fic3 réussit, celle de /data/fic3 a échoué.

Supprimer le répertoire /data.

```
# rm -rf /data
```

Exercice 3 : Création de volumes

Préalable :
Au du sein du répertoire de l'exercice, le fichier index.html est présent.
Faire du poste hôte un serveur NFS pour partager le répertoire /export/data.

```
# mkdir -p /export/data
# chmod  777  /export/data
# vi /etc/exports
/export/data  *(rw)
# systemctl start nfs
# firewall-cmd --add-service=nfs
# firewall-cmd --add-service=mountd
# firewall-cmd --add-service=rpc-bind
# exportfs
/export/data        <world>
```

Créer un fichier vide fic1 sous /export/data

```
# touch /export/data/fic1
```

Se positionner sur le répertoire de l'exercice et vérifier son contenu.

```
# ls -R /export/data/
```

Créer un volume nommé monsite.

```
# docker volume create monsite
```

Créer un volume nommé meslogs.

```
# docker volume create meslogs
```

Créer un volume nommé sauvegardes correspondant au montage NFS réalisé précédemment.

```
# docker volume create --driver local --opt type=nfs \
      --opt o=addr=adresse_ip_du_hote,rw --opt device=:/export/data \
  sauvegardes
```

Lister les volumes. Consulter le détails de ces volumes en les inspectant.

```
# docker volume ls
# docker volume inspect monsite
# docker volume inspect meslogs
# docker volume inspect sauvegardes
```

Exécuter un conteneur nommé web1 en mode démon d'une image nginx en mappant le port 91 du hôte avec le port 80 du conteneur et en mappant le volume monsite avec le répertoire /usr/share/nginx/html du conteneur.
Inspecter les informations du conteneur web1 concernant les volumes.

```
# docker run -d --name web1 -v monsite:/usr/share/nginx/html \
                            -p 91:80 nginx
# docker inspect web1
```

Consulter le site de web1.

```
# curl http://localhost:91
```

Copier le fichier index.html du répertoire de l'exercice au sein du répertoire du volume monsite.

```
# cp index.html /var/lib/docker/volumes/monsite/_data
```

Consulter le site de web1

```
# curl http://localhost:91
```

On obtient la nouvelle page.

Exécuter un nouveau conteneur nommé web2 en mode démon d'une image nginx en mappant le port 92 du hôte avec le port 80 du conteneur et en mappant le volume monsite avec le répertoire /usr/share/nginx/html du conteneur.
Consulter le site.

```
# docker run -d --name web2 -v monsite:/usr/share/nginx/html \
                            -p 92:80 nginx

# curl http://localhost:92
```

Les sites de web1 et web2 pointent sur la même page.

Supprimer les conteneurs web1 et web2.
Lister le contenu du répertoire du volume monsite.
Exécuter un conteneur nommé web1 en mode démon d'une image nginx en mappant le port 101 du hôte avec le port 80 du conteneur et en mappant le volume monsite avec le répertoire /usr/share/nginx/html du conteneur.
Consulter le site.

```
# docker rm -f web1 web2
# ls /var/lib/docker/volumes/monsite/_data
# docker run -d --name web1 -v monsite:/usr/share/nginx/html \
                            -p 101:80 nginx

# curl http://localhost:101
```

Les données sont toujours présente au sein du volume. Elles sont réutilisable par de nouveaux conteneurs.

Exécuter un conteneur nommé web2 en mode démon d'une image nginx en :

- mappant le port 102 du hôte avec le port 80 du conteneur,
- mappant le volume monsite avec le répertoire /usr/share/nginx/html du conteneur,
- mappant le volume meslogs avec le répertoire /var/log/nginx du conteneur,
- mappant le volume sauvegardes avec le répertoire /backup du conteneur.

```
# docker run -d --name web2 -p 102:80 -v monsite:/usr/share/nginx/html \
                -v meslogs:/var/log/nginx -v sauvegardes:/backup nginx
```

Inspecter les informations relatives aux volumes du conteneur web2.

```
# docker inspect web2
```

Consulter le site de web2.
Lister le contenu du répertoire /var/lib/docker/volumes/meslogs/_data. Vérifier qu'il y a le même contenu au sein de /var/log/nginx du conteneur web2.
Du conteneur web2, créer un fichier fic2 dans /backup.

Vérifier sur le serveur NFS que le fichier fic2 est présent au sein de /export/data.

```
# curl http://localhost:102
# ls /var/lib/docker/volumes/meslogs/_data
# docker exec web2 ls /var/log/nginx
# docker exec web2 ls /backup
ServeurNFS# ls /export/data
```

En utilisant l'option '--volumes-from', exécuter un conteneur en mode démon nommé web3 d'une image nginx qui utilise les mêmes volumes que le conteneur web2 et en mappant le port 103 du hôte au port 80 du conteneur.

```
# docker run -d --name web3 --volumes-from web2 -p 103:80 nginx
```

Consulter les volumes de web3 en inspectent les informations de ce conteneur.
Consulter le site de web3
Lister le contenu de /backup de web3.

```
# curl http://localhost:103
# docker exec web3 ls /backup
```

Supprimer les conteneurs web1, web2 et web.
Supprimer les conteneurs monsite, meslogs et sauvegardes.

```
# docker rm -f web1 web2 web3
# docker volume rm monsite meslogs sauvegardes
```

Exercice 4 : Création d'un registry privé

Créer un volume nommé stockage.

```
# docker volume create stockage
```

Créer un registry privé en exécutant un conteneur en mode démon avec les spécifications suivantes :

- utiliser l'image : registry:2
- nom du conteneur : depot
- mapper le port 5000 du hôte avec le port 5000 du conteneur.
- mapper le volume 'stockage' avec le répertoire /var/lib/registry du conteneur.
- prévoir un démarrage du conteneur avec le démarrage du hôte, ainsi qu'en cas d'arrêt impromptu.

```
# docker run -d --name depot -p 5000:5000 --restart=always \
                -v stockage:/var/lib/registry registry:2
```

Vérifier le fonctionnement et consulter les informations du conteneur depot.

```
# docker ps
# docker inspect depot
```

Le registry privé est prêt à être utilisé. Récupérer une image centos depuis le docker hub et donner lui un tag spécifique pour le registry privé 'depot' (localhost:5000/my_centos).

```
# docker pull centos
# docker tag centos:latest localhost:5000/my_centos
```

Vérifier que l'image a bien été taggée.
```
# docker images
```

Copier l'image dans le registry privé 'depot'.
```
# docker push localhost:5000/my_centos
```

Supprimer les images centos:latest et localhost:5000/my_centos du hôte local.
Vérifier.
```
# docker rmi centos localhost:5000/my_centos
# docker images
```

Exécuter un conteneur nommé cent1 en mode démon (-dit) à partir de l'image copiée dans le registry privé. Constater que l'image est téléchargée depuis le registry privé.
Lister les images présentes en local et lister les conteneurs en fonctionnement. L'image est-elle présente?
```
# docker run -dit --name cent1 localhost:5000/my_centos
# docker images
# docker ps
```
On constate la présence de l'image.

Supprimer le conteneur cent1 et l'image localhost:5000/my_centos.
```
# docker rm -f cent1
# docker rmi localhost:5000/my_centos
```

Supprimer le conteneur depot.
```
# docker rm -f depot
```

Recréer un registry privé nommé 'nouveau' avec les mêmes caractéristiques que précédemment.
```
# docker run -d --name nouveau -p 5000:5000 --restart=always \
                -v stockage:/var/lib/registry registry:2
```

Exécuter un conteneur nommé cent1 en mode démon (-dit) à partir de l'image copiée dans le registry privé. Constater que l'image est téléchargée depuis le registry privé.
Lister les images présentes en local et lister les conteneurs en fonctionnement. L'image est-elle présente?
```
# docker run -dit --name cent1 localhost:5000/my_centos
# docker images
# docker ps
```
On constate la présence de l'image.

Supprimer les conteneurs cent1 et nouveau.
Supprimer le volume stockage.
```
# docker rm -f cent1 nouveau
# docker volume rm stockage
```

Correction – Docker Compose

Exercice 1 : Installation de Docker Compose

Afin identifier la version de Docker Compose a installer, consulter le site :
https://github.com/docker/compose/releases

Installer Docker Compose (via curl).
Si la version est 1.23.0-rc1 :

```
# url_download="https://github.com/docker/compose/releases/download"

# curl -L ${url_download}/1.23.0-rc1/docker-compose-$(uname -s)-$(uname -m) \
       -o /usr/local/bin/docker-compose

# chmod  +x  /usr/local/bin/docker-compose
```

Vérifier par :

```
# docker-compose --version
# docker-compose version
```

Consulter l'aide de la commande :

```
# docker-compose --help
```

Exercice 2 : Un fichier de configuration de base

Se positionner dans le répertoire de l'exercice.
Il s'agit du premier exemple du support de cours. Veuillez à ne pas consulter le support !

Faire l'indentation du fichier docker-compose.yml pour pouvoir l'exécuter.
Rappel : docker-compose config pour valider la configuration du fichier.
docker-compose up -d pour l'exécuter.

```
# docker-compose config
# docker-compose up -d
```

Afficher l'aide de la commande docker-compose

```
# docker-compose --help
```

Utiliser la commande docker-compose pour :

- lister les images
- lister les services
- afficher le mappage du port 80 du service nginx
- consulter les logs
  - consulter le site : curl http://localhost
  - consulter les logs

arrêter les services. Vérifier. Redémarrer les services. Vérifier.

```
# docker-compose images
# docker-compose ps
# docker-compose port nginx 80
# docker-compose logs

# docker-compose ps
# docker-compose stop
# docker-compose ps
# docker-compose start
```

Arrêter les services avec suppression des conteneurs, réseaux et volumes les constituants.

```
# docker-compose down
```

Exercice 3 : Le projet wordpress

Se positionner dans le répertoire de l'exercice.
Il s'agit du projet du support de cours.

Exécuter le projet. Corriger l'erreur.
Exécuter à nouveau le projet.

```
Correction : retirer les tabulations.
# docker-compose up -d
```

Lister les services.

```
# docker-compose ps
```

Arrêter les services avec suppression des conteneurs, réseaux et volumes les constituants.
Redémarrer le projet en lui donnant un nom (utiliser votre prénom).
Lister les services.

```
# docker-compose down
# docker-compose -p jeanmarc up -d
# docker-compose -p jeanmarc ps
```

Tester en vous connectant à *http://localhost:8000* et en créant votre site. Puis connectez-vous.

Utiliser la commande docker pour afficher les réseaux et les volumes.

```
# docker network ls
# docker volume ls
```

Inspecter le réseau et le volume du projet.

```
# docker network inspect jeanmarc_default
# docker volume  inspect jeanmarc_bdd_data
```

Utiliser la commande docker-compose pour :

- lister les images
- lister les services
- afficher le mappage du port 80 du service wordpress
- consulter les logs

utiliser la sous commande top

```
# docker-compose -p jeanmarc images
# docker-compose -p jeanmarc ps
# docker-compose -p jeanmarc port wordpress 80
# docker-compose -p jeanmarc logs
# docker-compose -p jeanmarc top
```

Ouvrir une autre fenêtre terminale, consulter les événements (vous allez perdre la main).

```
# docker-compose -p jeanmarc events
```

Revenir à la première fenêtre terminale et générer de l'activité. Consulter à chaque étape le résultat généré sur les events.

Arrêter le service bdd.
Lister les services.
Démarrer le service bdd.

```
# docker-compose -p jeanmarc stop bdd
# docker-compose -p jeanmarc ps
# docker-compose -p jeanmarc start bdd
```

Fermer la fenêtre terminale des events.

Arrêter les services avec suppression des conteneurs, réseaux et volumes les constituants.
Qu'est-ce qui n'a pas été supprimé ?
Vérifier, et tout supprimer.

```
# docker-compose -p jeanmarc down

# docker-compose -p jeanmarc ps
# docker network ls              le réseau a été supprimé
# docker volume ls               le volume est toujours présent.
# docker system prune --volumes
# docker volume ls
```

Exercice 4 : Le projet nginx-php

Se positionner dans le répertoire de l'exercice.
Il s'agit du projet du support de cours.

Exécuter le projet..

```
# docker-compose up -d
```

Lister les services.

```
# docker-compose ps
```

Tester en vous connectant à *http://localhost*.

Utiliser la commande docker pour afficher les réseaux et les volumes.

```
# docker network ls
# docker volume ls
```

Utiliser la commande docker-compose pour :
- lister les images
- lister les services
- consulter les logs
- utiliser la sous commande top

```
# docker-compose images
# docker-compose ps
# docker-compose logs
# docker-compose top
```

Arrêter les services avec suppression des conteneurs, réseaux et volumes les constituants.
Vérifier.

```
# docker-compose  down

# docker-compose ps
# docker network ls              le réseau a été supprimé
```

Exercice 5 : Le projet LEMP

Un projet LEMP qui utilise nginx, php, mariadb et phpmyadmin.

Ce projet permet de développer les relations entre différents conteneurs.
Il met en évidence l'importance de l'organisation d'un projet (arborescence, logs, etc).
Il développe les différentes sous-commandes de docker-compose.

Se positionner dans le répertoire de l'exercice.

Arborescence du projet :

```
# tree  sitelemp
sitelemp/
├── docker-compose.yml
├── logs
│   ├── nginx-access.log
│   └── nginx-error.log
├── nginx
│   └── conf
│       └── default
└── site
    └── index.php
```

Attention lors de vos manipulations, il ne faut pas supprimer les deux fichiers nginx-access.log et nginx-error.log. Si cela arrive, recréez les :

```
# touch logs/nginx-access.log  logs/nginx-error.log
```

Le fichier docker-compose :

```
[root@mars sitelemp]# cat docker-compose.yml
version: '3'

services:
  web:
#    container_name: web
    image: tutum/nginx
    ports:
      - "80:80"
    links:
      - phpfpm
    volumes:
      - ./nginx/conf/default:/etc/nginx/sites-available/default
      - ./nginx/conf/default:/etc/nginx/sites-enabled/default
      - ./logs/nginx-access.log:/var/log/nginx/access.log
      - ./logs/nginx-error.log:/var/log/nginx/error.log

  phpfpm:
#    container_name: phpfpm
    image: php:fpm
    ports:
      - "9000:9000"
    volumes:
      - ./site:/usr/share/nginx/html

  mariadb:
#    container_name: mariadb
    image: mariadb
    environment:
      MYSQL_ROOT_PASSWORD: admin

  phpmyadmin:
#    container_name: phpmyadmin
    image: phpmyadmin/phpmyadmin
    restart: always
    links:
      - mariadb
    ports:
      - 8183:80
    environment:
      MYSQL_USERNAME: admin
      MYSQL_ROOT_PASSWORD: admin
      PMA_ARBITRARY: 1

[root@mars sitelemp]#
```

Les fichiers du répertoire logs sont des fichiers initialement vide.

```
[root@mars sitelemp]# cat site/index.php
<?php
phpinfo();
```

```
[root@mars sitelemp]# cat nginx/conf/default
server {
  listen 80;
  root /usr/share/nginx/html;
  index index.php index.html index.html;
  server_name 127.0.0.1;
  location / {
    try_files $uri /index.php$is_args$args;
  }
```

```
  location ~ \.php$ {
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    fastcgi_pass phpfpm:9000;
    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    include fastcgi_params;
  }
}
```

Valider le fichier de configuration :

```
[root@mars sitelemp]# docker-compose config
```

Exécuter le projet.

```
[root@mars sitelemp]# docker-compose  up  -d
Creating network "sitelemp_default" with the default driver
Pulling mariadb (mariadb:)...
latest: Pulling from library/mariadb
3d77ce4481b1: Pull complete
4f6a779d83f5: Pull complete
8c1d272f25d5: Pull complete
672dd5e0b768: Pull complete
84a7291b5996: Pull complete
92edc8e8d33d: Pull complete
f86a82067817: Pull complete
4ce9fcaa8405: Pull complete
59ec24cbcca5: Pull complete
86bfe0d99a20: Pull complete
4c8e32ff261f: Pull complete
Digest: sha256:21cbc4ff14023189c2004cd194976039318f31f2a0de11a3e2a4c85ff7c22fc1
Status: Downloaded newer image for mariadb:latest
Creating sitelemp_phpfpm_1  ... done
Creating sitelemp_mariadb_1 ... done
Creating sitelemp_web_1        ... done
Creating sitelemp_phpmyadmin_1 ... done
[root@mars sitelemp]#
```

http://127.0.0.1 on obtient la page d'informations de php (page.php)
http://127.0.0.1:8183 on obtient la page de phpmyadmin

Lister les services.

```
[root@mars sitelemp]# docker-compose  ps
Name                   Command                          State   Ports
-------------------------------------------------------------------------------------------
sitelemp_mariadb_1     docker-entrypoint.sh mysqld      Up      3306/tcp

sitelemp_phpfpm_1      docker-php-entrypoint php- fpm   Up      0.0.0.0:9000->9000/tcp

sitelemp_phpmyadmin_1  /run.sh phpmyadmin               Up      0.0.0.0:8183->80/tcp,
                                                                9000/tcp
sitelemp_web_1         /usr/sbin/nginx                  Up      0.0.0.0:80->80/tcp
[root@mars sitelemp]#
```

Inspecter le conteneur sitelemp_web_1_* .

```
[root@mars sitelemp]# docker inspect sitelemp_web_1_....
```

Lister les réseaux avec la commande docker.

```
[root@mars sitelemp]# docker network ls
```

Utiliser la sous commande exec de docker-compose pour exécuter sh au conteneur mariadb.
Puis taper les commandes suivantes : mysql -u root -p
mot de passe : admin
show databases ;
exit
exit

```
[root@mars sitelemp]# docker-compose exec mariadb sh
# mysql -u root -p
Enter password:                  --> admin
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 8
Server version: 10.2.15-MariaDB-10.2.15+maria~jessie mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
3 rows in set (0.04 sec)

MariaDB [(none)]> exit
Bye
# exit
[root@mars sitelemp]#
```

Lister les images.

```
[root@mars sitelemp]# docker-compose images
Container              Repository              Tag      Image Id       Size
-----------------------------------------------------------------------------------
sitelemp_mariadb_1     mariadb                 latest   447a28508139   383 MB
sitelemp_phpfpm_1      php                     fpm      0a757334c1f6   351 MB
sitelemp_phpmyadmin_1  phpmyadmin/phpmyadmin   latest   4bdc31ab2ded   156 MB
sitelemp_web_1         tutum/nginx             latest   a2e9b71ed366   197 MB
[root@mars sitelemp]#
```

Consulter les deux fichiers de logs logs/nginx-access.log et logs/nginx-error.log.

```
[root@mars sitelemp]# cat logs/nginx-error.log
2018/05/26 21:53:16 [error] 5#5: *1 open() "/usr/share/nginx/html/index.php" failed (2:
No such file or directory), client: 172.18.0.1, server: localhost, request: "GET
/index.php HTTP/1.1", host: "127.0.0.1"
2018/05/26 21:59:04 [error] 6#6: *2 open() "/usr/share/nginx/html/favicon.ico" failed (2:
No such file or directory), client: 172.18.0.1, server: localhost, request: "GET
/favicon.ico HTTP/1.1", host: "172.18.0.1"
 . . .
```

```
[root@mars sitelemp]# cat logs/nginx-access.log
172.18.0.1 - - [26/May/2018:21:53:16 +0000] "GET /index.php HTTP/1.1" 404 170 "-"
"Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
172.18.0.1 - - [26/May/2018:21:59:04 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0
(X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
 . . .
```

Consulter les logs via la commande docker-compose.

```
[root@mars sitelemp]# docker-compose logs
Attaching to sitelemp_phpmyadmin_1, sitelemp_web_1, sitelemp_phpfpm_1, sitelemp_mariadb_1
phpfpm_1      | [26-May-2018 22:52:36] NOTICE: fpm is running, pid 1
phpfpm_1      | [26-May-2018 22:52:36] NOTICE: ready to handle connections
phpfpm_1      | 172.18.0.4 -  26/May/2018:23:08:49 +0000 "GET /index.php" 200
mariadb_1     | Initializing database
 . . .
```

Utiliser la sous commande top de docker-compose.

```
[root@mars sitelemp]# docker-compose top
sitelemp_mariadb_1
  UID      PID    PPID    C   STIME   TTY     TIME      CMD
-----------------------------------------------------------
polkitd   22286   22264   0   00:51   ?     00:00:09   mysqld

sitelemp_phpfpm_1
UID     PID    PPID    C   STIME   TTY     TIME                        CMD
-------------------------------------------------------------------------------------
----
root   22298   22277   0   00:51   ?     00:00:00   php-fpm: master process
                                                    (/usr/local/etc/php-fpm.conf)
33     22424   22298   0   00:52   ?     00:00:00   php-fpm: pool www
33     22425   22298   0   00:52   ?     00:00:00   php-fpm: pool www

sitelemp_phpmyadmin_1
  UID       PID    PPID    C   STIME   TTY     TIME                         CMD
-------------------------------------------------------------------------------------
----
root       22485   22472   0   00:52   ?     00:00:02   /usr/bin/python2
             /usr/bin/supervisord --nodaemon --configuration=/etc/supervisord.conf
             --loglevel=info
nfsnobo+   22681   22485   0   00:52   ?     00:00:00   php-fpm: master process
                                                        (/etc/php-fpm.conf)
root       22682   22485   0   00:52   ?     00:00:00   nginx: master process nginx -c
                                                        /etc/nginx.conf
nfsnobo+   22683   22682   0   00:52   ?     00:00:00   nginx: worker process
nfsnobo+   22684   22682   0   00:52   ?     00:00:00   nginx: worker process
nfsnobo+   22685   22682   0   00:52   ?     00:00:00   nginx: worker process
nfsnobo+   22686   22682   0   00:52   ?     00:00:00   nginx: worker process
nfsnobo+   22687   22681   0   00:53   ?     00:00:00   php-fpm: pool www
nfsnobo+   22688   22681   0   00:53   ?     00:00:00   php-fpm: pool www

sitelemp_web_1
UID     PID    PPID    C   STIME   TTY     TIME                        CMD
-------------------------------------------------------------------------------------
root   22457   22438   0   00:52   ?     00:00:00   nginx: master process /usr/sbin/nginx
33     22658   22457   0   00:52   ?     00:00:00   nginx: worker process
33     22659   22457   0   00:52   ?     00:00:00   nginx: worker process
```

```
33     22660   22457   0   00:52   ?     00:00:00   nginx: worker process
33     22661   22457   0   00:52   ?     00:00:00   nginx: worker process
```

Arrêter le projet.

```
[root@mars sitelemp]# docker-compose down
Stopping sitelemp_phpmyadmin_1 ... done
Stopping sitelemp_web_1        ... done
Stopping sitelemp_phpfpm_1     ... done
Stopping sitelemp_mariadb_1    ... done
Removing sitelemp_phpmyadmin_1 ... done
Removing sitelemp_web_1        ... done
Removing sitelemp_phpfpm_1     ... done
Removing sitelemp_mariadb_1    ... done
Removing network sitelemp_default
[root@mars sitelemp]#
```

Vérifier que le site ne fonctionne plus.
Relancer le projet.

```
[root@mars sitelemp]# docker-compose up -d
Creating network "sitelemp_default" with the default driver
Creating sitelemp_mariadb_1 ... done
Creating sitelemp_phpfpm_1     ... done
Creating sitelemp_phpmyadmin_1 ... done
Creating sitelemp_web_1        ... done
[root@mars sitelemp]#
```

Vérifier que le site fonctionne à nouveau.

Arrêter le projet et supprimer les volumes.

```
[root@mars sitelemp]# docker-compose down
Stopping sitelemp_web_1        ... done
 . . .
Removing sitelemp_web_1        ... done
 . . .
Removing network sitelemp_default
[root@mars sitelemp]#
```

```
[root@mars sitelemp]# docker system prune --volumes
WARNING! This will remove:
        - all stopped containers
        - all networks not used by at least one container
        - all volumes not used by at least one container
        - all dangling images
        - all build cache
Are you sure you want to continue? [y/N] y
Deleted Volumes:
2d0290234bede8245fb668d73d1fee268761e9562e04476aac5537004437f942
8b3447eb999811e1395d5d020f8291eb05c250e5f36edcfb1436d02a5c8ef334

Total reclaimed space: 251.6MB
[root@mars sitelemp]#
```

Correction – Docker Machine

Préalable : Installation de Docker Toolbox.

Du site de Docker, télécharger 'Docker Toolbox' sur votre poste Windows et installez le.
Démarrer 'Docker Quickstart Terminal' pour réaliser les exercices de ce module.

Exercice 1 : Déploiement de machines

Créer quatre machines virtuelles 'Oracle VirtualBox' avec la commande docker-machine.
Elles auront les noms suivants : master, worker1, worker2 et worrker3.

```
# docker-machine create -d virtualbox master
# docker-machine create -d virtualbox worker1
# docker-machine create -d virtualbox worker2
# docker-machine create -d virtualbox worker3
```

Vérifier qu'elles fonctionnent.

```
# docker-machine ls
```

Exercice 2 : Fonctionnalités de docker-machine

Créer une machine virtuelle 'Oracle VirtualBox' via la commande docker-machine nommée 'perso'.

```
# docker-machine create -d virtualbox perso
# docker-machine ls
```

Connectez-vous par ssh à la machine 'perso'et taper les commandes suivantes :

id, pwd, ls, hostname, ip a.

vérifier que le démon dockerd fonctionne (ps -ef | grep -i dockerd).

Se déconnecter par 'exit'.

```
# docker-machine ssh perso
     id     ls     hostname     ip a     ps -ef | grep -i dockerd
# exit
```

Inspecter les informations détaillées de la machine 'perso'.

```
# docker-machine inspect perso
```

Afficher l'adresse IP de la machine 'perso', puis son url et son status.

```
# docker-machine ip perso
# docker-machine url perso
# docker-machine status perso
```

Identifier le nom de la machine active.
Prendre l'environnement de la machine 'perso'.
Identifier le nom de la machine active.
Exécuter un conteneur nommé alp1 avec l'option '-dit' d'une image alpine.
Lister les images, puis les conteneurs en cours de fonctionnement.

```
# docker-machine active
# eval $(docker-machine env perso)
# docker-machine active
# docker run -dit --name alp1 alpine
# docker images
# docker ps
```

Prendre l'environnement de la machine 'default'.
Identifier le nom de la machine active.
Lister les images, puis les conteneurs en cours de fonctionnement.
Prendre l'environnement de la machine 'perso'.
Lister les images, puis les conteneurs en cours de fonctionnement.
Supprimer le conteneur alp1 et l'image alpine.

```
# eval $(docker-machine env default)
# docker-machine active
# docker images
# docker ps
# eval $(docker-machine env perso)
# docker-machine active
# docker images
# docker ps
# docker rm -f alp1
# docker rmi alpine
```

Prendre l'environnement de la machine 'default'.
Redémarrer la machine 'perso'.
Vérifier qu'elle a bien redémarrée.

```
# eval $(docker-machine env default)
# docker-machine active
# docker-machine restart perso
# docker-machine ls
```

Arrêter la machine 'perso'. Vérifier.
Supprimer la machine 'perso'.

```
# docker-machine stop perso
# docker-machine ls
# docker-machine rm perso
```

Exercice 3 : Déploiement customisé

Créer une machine virtuelle 'Oracle VirtualBox' via la commande docker-machine avec les caractéristiques suivantes : le nom est mavm, 1 Giga de RAM (1024 Mégas) et un disque de 2 Gigas (2048 Mégas).
Aidez-vous des aides de la commande pour identifier les options adéquates.

```
# docker-machine create -d virtualbox --virtualbox-memory 1024 \
                        --virtualbox-disk-size 2048  mavm
```

Utiliser les options de la commande docker-machine pour afficher des informations sur cette machine virtuelle.

```
# docker-machine {status|ls|inspect} mavm
```

Se connecter à la machine virtuelle et vérifier sa configuration : le hostname, la taille de la mémoire, la taille du disque et que le service de docker engine fonctionne. Puis se déconnecter.

```
# docker-machine ssh mavm
# hostname
# df -h
# free -h
# ps -ef | grep -i docker
# exit
```

Supprimer cette machine virtuelle.

```
# docker-machine stop mavm
# docker-machine rm   mavm
```

Correction – Docker Swarm

Préalable :
Avoir les quatre machines virtuelles créées lors des exercices sur le thème de docker-machine :
master, worker1, worker2 et worker3.

Exercice 1 : Activation d'un cluster swarm

Afficher l'état de swarm par : docker info

```
$ docker info
$ docker info -f '{{.Swarm}}'
```

Se connecter à la machine master.
Déclarer la machine master comme le leader du cluster.

```
$ docker-machine ssh master
docker@master:~$ docker  swarm  init  --advertise-addr ip_master
```

Afficher l'état de swarm par : docker info

```
$ docker info
```

Lister les nœuds du cluster.

```
docker@master:~$ docker node ls
```

Se déconnecter du master.

```
docker@master:~$ exit
```

Se connecter à la machine worker1.
Intégrer la machine worker1 au cluster.
Afficher l'état de swarm par : docker info
Se déconnecter de la machine worker1.

```
$ docker-machine  ssh  node1
docker@node1:~$  docker  swarm  join  --token SWMTKN-1-
26xfub8oviboh5nwcfrq1plhr6a2zdeyqjv23v13o3qgsp8fqn-
f4fokyffk8095kced59ybuozm  192.168.99.101:2377

docker@node1:~$ docker  info
docker@node1:~$ exit
```

Refaire les mêmes manipulations pour intégrer au cluster : worker2 et worker3.

Prendre l'environnement du master par :

```
$ eval "$(docker-machine env master)"
```

Afficher l'état de swarm par : docker info

```
$ docker info
```

Lister les nœuds du cluster.

```
docker@master:~$ docker node ls
```

Exercice 2 : Management de services swarm

Lister les réseaux. Que remarquez-vous ?

```
docker@master:~$ docker network ls
```

Deux réseaux ont été créés suite à l'activation du cluster swarm.

Inspecter le réseau ingress.

```
docker@master:~$ docker inspect ingress
```

Démarrer un service nginx nommé web dans le cluster (utiliser '-p 80:80' pour mapper les ports).

```
docker@master:~$ docker service create --name web -p 80:80 nginx
```

Lister les services (ls).
Lister les instances du service web (ps).
Tester le site via curl http://adresse_ip_master
Tester le site via curl avec les adresses des workers.
Consulter les logs du service web.
Inspecter le service web (avec et sans l'option –pretty).

```
$ docker service ls
$ docker service ps web
$ curl http://adresse_ip_master       ou des workers.
$ docker service logs web
$ docker service inspect web
$ docker service inspect --pretty web
```

Supprimer le service web.

```
$ docker service rm web
```

Démarrer 8 services nginx nommé web dans le cluster (utiliser '-p 80:80' pour mapper les ports).
Utiliser l'option '--replicas N' de 'docker service' pou lancer N instances.

```
$ docker service create --name web -p 80:80 --replicas 8 nginx
```

Lister les services.
Lister les instances du service web.
Tester le site via curl http://adresse_ip_master_ou_worker
Consulter les logs du service web.
Inspecter le service web (avec l'option –pretty).

```
$ docker service ls
$ docker service ps web
$ curl http://adresse_ip_master       ou des workers.
$ docker service logs web
$ docker service inspect --pretty web
```

Monter à 12 instances du service web.
Lister les services. Lister les instances du service web.

```
$ docker service scale web=12
$ docker service ls
$ docker service ps web
```

Descendre à 5 instances du service web.
Lister les services. Lister les instances du service web.

```
$ docker service scale web=5
$ docker service ls
$ docker service ps web
```

Revenir à 8 instances du service web.
Lister les services. Lister les instances du service web.

```
$ docker service scale web=8
$ docker service ls
$ docker service ps web
```

Vérifier le nombre d'instance qui fonctionne sur worker3.
Arrêter la machine worker3 (docker-machine stop worker3).
Lister les services. Lister les instances du service web.

```
$ docker service ps web
$ docker-machine stop worker3
$ docker service ls
$ docker service ps web
```

Redémarrer la machine worker3.
Lister les services. Lister les instances du service web.

```
$ docker-machine start worker3
$ docker service ls
$ docker service ps web
```

Supprimer tous les services web.

```
$ docker service rm web
```

Exercice 3 : Management des masters swarm

Lister les nœuds du cluster.

```
$ docker node ls
```

Promouvoir les worker2 et worker3 en master.

```
$ docker node promote worker2
$ docker node promote worker3
```

Lister les nœuds du cluster, puis afficher les informations du cluster.

```
$ docker node ls
$ docker info
```

Démarrer 8 services nginx nommé web dans le cluster (utiliser '-p 80:80' pour mapper les ports).
Utiliser l'option '--replicas N' de 'docker service' pou lancer N instances.

```
$ docker service create --name web -p 80:80 --replicas 8 nginx
```

Lister les services.
Lister les instances du service web.
Inspecter le service web (avec l'option –pretty).

```
$ docker service ls
$ docker service ps web
$ docker service inspect --pretty web
```

Arrêter le master. Vérifier avec docker-machine ls.

```
$ docker-machine stop master
$ docker-machine ls
```

Prendre l'environnement de worker3 (ou se connecter worker3).
Lister les nœuds du cluster, puis afficher les informations du cluster.
Lister les services. Lister les instances du service web.

```
$ eval $(docker-machine env worker3)
$ docker node ls
$ docker info
$ docker service ls
$ docker service ps web
```

Redémarrer le master. Vérifier.
Lister les instances du service web.

```
$ docker-machine start master
$ docker-machine ls
$ docker node ls
$ docker service ps web
```

Monter à 12 instances du service web.
Lister les instances du service web.

```
$ docker service scale web=12
$ docker service ps web
```

Supprimer le service web.

```
$ docker service rm web
```

Exercice 4 : Service répliqué et service global

--mode replicated : pour un service répliqué (valeur par défaut).
Le nombre de services est assuré même en cas de défaillance d'une machine.
--mode global : pour un service global.
Un service par machine du cluster.

Arrêter la machine worker1.
Lister les instances du service web.

```
$ docker-machine stop worker1
$ docker-machine ls
```

Démarrer un service nommé web de l'image nginx en mode global (en mappant les ports).

```
$ docker service create --mode global --name web -p 80:80 nginx
```

Lister les services. Lister les instances du service web.

```
$ docker service ls
$ docker service ps web
```

Démarrer la machine worker1.
Lister les services. Lister les instances du service web.

```
$ docker-machine start worker1
$ docker-machine ls
$ docker service ls
$ docker service ps web
```

Démarrer 4 instances d'un service nommé alp1 de l'image alpine en mode répliqué qui exécute la commande 'ping 8.8.8.8'.

```
$ docker service create --name alp1 --replicas 4 alpine ping 8.8.8.8
```

Lister les services. Lister les instances du service web et de alp1.

```
$ docker service ls
$ docker service ps web
$ docker service ps alp1
```

Arrêter la machine worker1.
Lister les services. Lister les instances du service web et de alp1.

```
$ docker-machine stop worker1
$ docker-machine ls
$ docker service ps web
$ docker service ps alp1
```

Démarrer la machine worker1.
Lister les services. Lister les instances du service web.

```
$ docker-machine start worker1
$ docker-machine ls
$ docker service ls
$ docker service ps web
```

Supprimer les service web et alp1.

```
$ docker service rm web alp1
```