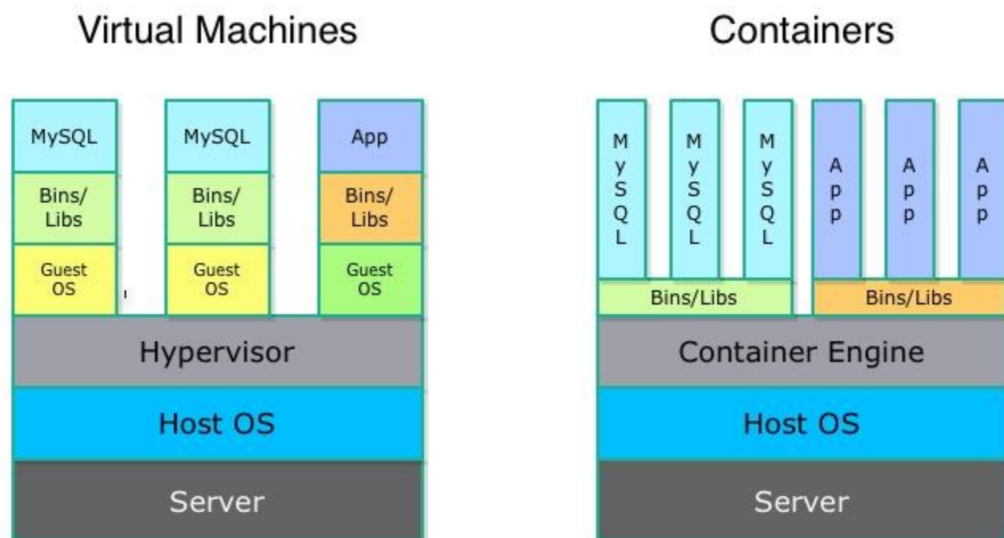
Exercice sur les Dockers

Les parties composant les Dockers:

1. docker daemon: est un démon qui gère les dockers (les conteneurs LXC) qui tournent sur la machine hôte
2. docker CLI: une série de commandes utiles pour interagir avec le démon
3. docker image index: est une base d'images binaires (publiques ou privées)

Les éléments relatifs aux Dockers:

1. docker containers: des répertoires contenant l'application.
2. docker images: images binaires ou OS de base (ex. centos, Ubuntu, etc.)
3. Dockerfiles: des scripts qui automatisent la manipulation d'images.



Différence entre VM et Conteneurs
http://patg.net/containers,virtualization,docker/2014/06/05/docker-intro/

1. Manipulation de conteneurs

La commande suivante devrait vous permettre de télécharger la première fois une image légère centos à partir du Cloud (repository), la démarrer dans un conteneur avant de lancer une session shell:

```
$ sudo docker run -i -t centos /bin/bash
```

Conteneur interactif:

```
$ sudo docker run -t -i centos /bin/bash

root@af8bae53bdd3:/#
```

-i : utilisation interactive
-t : affichage sur terminal

Exemple:

```
root@af8bae53bdd3:/# pwd

/

root@af8bae53bdd3:/# ls

bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp
usr var
```

Pour quitter:

```
root@af8bae53bdd3:/# exit
```

Autre exemple:
Un conteneur lancé en arrière plan (-d) qui lance un script. Cette commande retourne l'ID du conteneur.

```
$ sudo docker run -d centos /bin/sh -c "while true; do echo hello world;
sleep 1; done"

1e5535038e285177d5214659a068137486f96ee5c2e85a4ac52dc83f2ebe4147 is the
container ID provided by the docker daemon
```

Quelques commandes de manipulation de conteneurs

a. Affiche la version du Docker et donne en plus la version du langage Go utilisé :

```
$ sudo docker version
```

b. Affiche la liste des containers :

```
$ sudo docker ps

CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS
PORTS NAMES
```

```
1e5535038e28  ubuntu:14.04  /bin/sh -c 'while tr  2 minutes ago  Up 1 minute
insane_babbage
```

Affiche la liste des conteneurs déjà lances:
`$ sudo docker ps -l`

Affiche la liste des conteneurs actifs:
`$ sudo docker ps`

Affcihe tous les conteneurs y compris ceux qui sont stoppés ou terminés:
`$ sudo docker ps -a`

c. Affiche les logs :

```
$ sudo docker logs insane_babbage

hello world

hello world

hello world

. . .
```

d. Stoppe un container:

```
$ sudo docker stop insane_babbage
```

e. Voir ce que le client Docker peut faire:

```
$ sudo docker
```

Voir la liste de commandes:

```
Commands:

     attach    Attach to a running container

     build     Build an image from a Dockerfile

     commit    Create a new image from a container's changes

. . .
```

On peut passer l'option `--help`, exemple :

```
$ sudo docker attach --help
```

Si on veut ré-accéder à un container, on commence par le relancer le conteneur en précisant son nom et ensuite on l'attache :

```
$ sudo docker start insane_babbage

$ sudo docker attach insane_babbage
```

2. Manipulation d'images binaires

La commande suivante affiche la liste des images binaires dont nous disposons en local :

```
$ sudo docker images

REPOSITORY        TAG      IMAGE ID      CREATED      VIRTUAL SIZE

training/webapp   latest   fc77f57ad303  2 weeks ago  280.5 MB

ubuntu            13.10    5e019ab7bf6d  3 weeks ago  180 MB

ubuntu            saucy    5e019ab7bf6d  4 weeks ago  180 MB

ubuntu            12.04    74fe38d11401  4 weeks ago  209.6 MB
```

On peut recherche une image sur le Cloud (Docker repository) :

```
$ sudo docker search opensuse
```

Pour obtenir une nouvelle image, on peut en pré-charger une comme par exemple opensuse :

```
$ sudo docker pull opensuse

Pulling repository opensuse

b7de3133ff98: Pulling dependent layers

5cc9e91966f7: Pulling fs layer

511136ea3c5a: Download complete

ef52fb1fe610: Download complete
```

```
. . .
```

Ainsi lors de cette commande, il ne sera pas nécessaire de faire du téléchargement :

```
$ sudo docker run -t -i opensuse /bin/bash

bash-4.1#
```

3. Suivre les indications données dans ce lien pour vous enregistrer et créer votre propre espace de stockage (repository ou Hub) sur le Cloud : https://hub.docker.com/

Une manière de créer un compte est de lancer la commande suivante :

```
$ sudo docker login
```

Un fichier .dockercfg contient les authentifications.

La création du repository personnel nécessite une adresse mail, un login (par exemple Samia) et un mot de passe.

4. Créer un container avec une image binaire Opensuse. Enrichir Opensuse en installant une application. Par exemple un programme C jouant le rôle de serveur et communiquant via des sockets avec un Client C.

Il y a beaucoup d'exemples de programmes en C sur Internet, voir par exemple ici : http://www.binarytides.com/server-client-example-c-sockets-linux/

Sauvegarder la nouvelle image dans votre Hub personnel.

```
sudo docker ps -l

sudo docker run -t -i opensuse /bin/bash
```

On peut identifier chaque container avec container ID qui est une succession de chiffres.

L'image opensuse est très basique et ne contient ni compilateur *gcc* ni éditeur de texte, il faut donc les installer pour pouvoir modifier cette image et y créer un programme serveur en langage C par exemple.

Pour installer le compilateur *gcc* sur opensuse il faut utiliser la commande suivante :

```
zypper install gcc
```

Pour pouvoir éditer des fichiers sur opensuse, il faut installer *vi* ou *vim* :

```
zypper install vi

zypper install vim
```

Sur opensuse créons un fichier contenant le code du serveur en C (server.c) qui sera compilé et qui va générer un exécutable (server.o) avec la commande suivante:

```
gcc server.c -o server.o
```

Une fois toutes les modifications de opensuse terminées, on exécute la commande *exit* pour sortir du container. Cependant ceci ne permet pas de sauvegarder directement le contenu du container. Pour cela, il faut créer une image à partir de ce container. Nous allons appeler cette nouvelle image opensuse:v2.
Toute modification de l'image binaire associée à un container (de numéro par exemple 3f5907e5b0f) requiert un commit :

```
sudo docker commit -m="Added server" -a="Samia" 3f5907e5b0f samia/opensuse:v2
```

Pour sauvegarder la nouvelle image dans le Hub personnel (Cloud), il faut utiliser la commande push. Cette opération exige de fournir le login, mot de passe et email utilisés sur le Cloud.

```
sudo docker push samia/opensuse
```

Reconnectez-vous à votre repository pour vérifier s'il y a bien votre opensuse.

5. Supprimer l'image binaire localement de la machine.

La suppression se fait avec la commande

```
sudo docker rmi -f opensuse:v2
```

6. Ramener votre image du Hub et tester l'application intégrée dans l'image en l'appelant via un Client qui se lance sous Ubuntu. Ceci nécessitera un mapping entre l'adresse IP locale de l'image opensuse avec l'adresse IP accessible via Ubuntu.

Pour charger l'image à partir du repository personnel il faut utiliser la commande suivante:

```
sudo docker run -i -t samia/opensuse /bin/bash
```

Pour lancer le serveur sur l'image ainsi démarrée, il faut faire un mapping entre l'adresse IP de la machine hôte Ubuntu et l'adresse IP de l'image opensuse. Ceci permettra de faire suivre les requêtes du Client vers le serveur lancé dans le container. L'option –p est utilisée pour désigner les ports et adresses IP.
Dans notre exemple, nous lions le port 5013 du container au port 5013 de la machine locale (adresse ip localhost : 127.0.0.1).

```
sudo docker run -i -t -p 127.0.0.1:5013:5013 samia/opensuse:v2 ./server.o
```

Le serveur se met alors en attente d'une connexion du client.
On peut donc lancer le client sur Ubuntu avec la commande :

```
./client.o
```

Annexes: http://www.binarytides.com/server-client-example-c-sockets-linux/
http://cedric.cnam.fr/~bouzefra/cours/serveur.c

```
/*
    C socket server example
*/

#include<stdio.h>
#include<string.h>    //strlen
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr
#include<unistd.h>    //write

int main(int argc , char *argv[])
{
    int socket_desc , client_sock , c , read_size;
    struct sockaddr_in server , client;
    char client_message[2000];

    //Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1)
    {
        printf("Could not create socket");
    }
    puts("Socket created");

    //Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(5013);

    //Bind
    if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
    {
        //print the error message
        perror("bind failed. Error");
        return 1;
    }
    puts("bind done");

    //Listen
    listen(socket_desc , 3);

    //Accept and incoming connection
    puts("Waiting for incoming connections...");
    c = sizeof(struct sockaddr_in);

    //accept connection from an incoming client
    client_sock = accept(socket_desc, (struct sockaddr *)&client,
(socklen_t*)&c);
    if (client_sock < 0)
    {
        perror("accept failed");
        return 1;
    }
    puts("Connection accepted");

    //Receive a message from client
    while( (read_size = recv(client_sock , client_message , 2000 , 0)) > 0 )
    {
        //Send the message back to client
```

```
        write(client_sock , client_message , strlen(client_message));
    }

    if(read_size == 0)
    {
        puts("Client disconnected");
        fflush(stdout);
    }
    else if(read_size == -1)
    {
        perror("recv failed");
    }

    return 0;
}
```

http://cedric.cnam.fr/~bouzefra/cours/client.c

```
/*
    C ECHO client example using sockets
*/
#include<stdio.h> //printf
#include<string.h>    //strlen
#include<sys/socket.h>    //socket
#include<arpa/inet.h> //inet_addr

int main(int argc , char *argv[])
{
    int sock;
    struct sockaddr_in server;
    char message[1000] , server_reply[2000];

    //Create socket
    sock = socket(AF_INET , SOCK_STREAM , 0);
    if (sock == -1)
    {
        printf("Could not create socket");
    }
    puts("Socket created");

    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_family = AF_INET;
    server.sin_port = htons(5013);

    //Connect to remote server
    if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
    {
        perror("connect failed. Error");
        return 1;
    }

    puts("Connected\n");

    //keep communicating with server
    while(1)
    {
        printf("Enter message : ");
        scanf("%s" , message);

        //Send some data
```

```
        if( send(sock , message , strlen(message) , 0) < 0)
        {
            puts("Send failed");
            return 1;
        }

        //Receive a reply from the server
        if( recv(sock , server_reply , 2000 , 0) < 0)
        {
            puts("recv failed");
            break;
        }

        puts("Server reply :");
        puts(server_reply);
    }

    close(sock);
    return 0;
}
```