R4.08 - TD 2 - Réseau

Introduction

La conteneurisation est très utilisée dans les architectures micro-services.

Une architecture micro-services est une architecture logicielle dans laquelle les fonctions sont découpées en petites briques, les services : un serveur Web, une BDD, etc.

Très souvent, les services communiquent entre eux par le réseau.

Par sa conception, Docker est naturellement un outil très bien adapté pour construire de tels environnements micro-services.

Exercice 1 - Réseau interne

Lancez une commande (sur l'hôte, pas dans un conteneur) :

```
php -S localhost:1234
```

Vous devez le savoir, cette commande permet de lancer **PHP** en mode serveur Web. Laissez-le tourner dans le terminal.

Dans un second terminal, lancez maintenant une commande :

```
netstat -nate
```

Cette commande permet de lister toutes connexions actives sur votre machine. C'est une commande Linux, elle n'est pas spécifique à Docker.

Cherchez-y une ligne **`127.0.0.1:1234`** (**`127.0.0.1`** est l'IP de **`localhost`**). Vous devez repérer cette connexion dans la liste. Il est aussi possible que cette ligne soit en IPv6, sous la forme **`::1:1234`**

Arrêtez votre commande **`php`** et refaites la même expérience en lançant, cette fois-ci, la commande **`php`** dans un conteneur sur la base d'une image **`r408-php`** :

```
docker run --rm -ti r408-php php -S localhost:1234
```

tout en continuant de lancer la commande **`netstat`** sur votre machine hôte.

Trouvez-vous la ligne **`127.0.0.1:1234`** (ou sa forme IPv6 **`::1:1234`**) cette fois-ci ?

Un conteneur isole son contenu (fichier et processus). Il en va de même pour sa partie réseau.

Exercice 2 - nginx

On vient de voir qu'un programme proposant des services via le réseau mais s'exécutant dans un conteneur, n'est pas accessible depuis l'extérieur du conteneur. A ce stade, ça limite donc grandement son intérêt, surtout dans une infrastructure micro-services où tous les services sont isolés dans plusieurs conteneurs et non pas dans un seul !

Heureusement, il existe une solution.

Pour autoriser le monde extérieur à se connecter à un service réseau tournant dans un conteneur, il faut établir une sorte de passerelle entre l'extérieur et l'intérieur du conteneur.

Pour ce faire, à la création du conteneur, il faut ajouter une option de mappage de port.

Le mappage d'un port se fait par l'ajout d'une option **`-p`**, ainsi :

> **`docker container run -p port_sur_hôte:port_dans_conteneur ...`**

Le mappage ne peut se faire qu'à la création du conteneur.

Ce mappage agit comme un aiguilleur. Toute connexion arrivant sur l'hôte sur son port **`port_sur_hôte`** sera aiguillée vers le port **`port_dans_conteneur`** DANS le conteneur.

Testez avec une image **`r408-nginx`** (tag **`latest`**) en créant un conteneur avec une redirection du port **`1234`** de votre machine hôte vers le port adéquat dans le conteneur. Si vous vous demandez quel est ce port "adéquat", deux façons de le deviner en sachant que :

1) **`nginx`** est un serveur Web, vous devriez donc avoir une idée du port de travail standard d'un serveur Web
2) si vous n'avez vraiment pas d'idée, c'est dommage mais allez voir sur le Docker Hub… ou demandez à qui vous savez (non, ce n'est pas Google, il doit être en face de vous)

De chez vous ou sur votre ordinateur personnel, utilisez simplement l'image **`nginx:1.23`**

Une fois votre conteneur en place et actif, dans votre navigateur Web, rendez-vous sur **`http://localhost:1234`**, vous devriez avoir une page **Welcome to nginx!**

Exercice 3 - Un mini site Web

Maintenant que vous avez un serveur Web opérationnel (**Exercice 2**), vous allez créer une page **`index.html`** avec un petit contenu de votre choix, associé à un peu de CSS dans un fichier **`style.css`**.

Cette page doit être servie par un serveur **`nginx`**.

La racine du serveur **`nginx`** (où sont les fichiers du site) est **`/usr/share/nginx/html`**.

Nous partirons aussi du principe que le conteneur peut-être supprimé mais que votre page et son fichier de style doivent être préservés en cas de suppression du conteneur.

Quelle est la solution ?

Adaptez le lancement du conteneur pour mettre en œuvre cette solution.

Testez dans votre navigateur que **`http://localhost:1234`** affiche bien votre page maintenant.

Vérifiez aussi que vous pouvez modifier facilement votre page et son style dans VSC et que ces modifications sont immédiatement prises en compte par un rafraîchissement de la page dans le navigateur.

Exercice 4 - Création d'un réseau

Docker a la capacité de créer des réseaux virtuels privés (VPN) locaux, qui permettent à des conteneurs de se voir entre eux, de façon cloisonnée et isolée de l'hôte.

On a vu que les conteneurs sont un peu comme des machines virtuelles dans une machine physique. Les réseaux Docker sont donc un peu comme des câbles virtuels branchés entre certaines de ces "machines virtuelles" que sont les conteneurs.

Vous décidez quels conteneurs sont reliés à quels réseaux, en fonction de vos besoins.

L'idée maîtresse étant toujours de cloisonner de tout ce qu'il est possible de cloisonner ! On ne met deux conteneurs sur un même réseau que s'ils ont une raison de se voir, et ont donc un besoin de parler entre eux.

Créer un réseau dans Docker est indépendant de la création d'un conteneur. Un réseau peut exister, même si aucun conteneur n'est "branché" dessus. C'est comme dans le monde physique : un réseau peut exister même si aucune machine physique n'y est connectée. La borne wifi chez vous existe même si aucun appareil n'y est relié.

Ainsi, la création d'un réseau Docker doit se faire avant tout rattachement d'un conteneur à ce réseau.

Voici la syntaxe pour créer un réseau :

```
docker network create nom_du_reseau
```

Créez un réseau nommé **mon_rezo**.

Pour afficher la liste des réseaux Docker présents sur votre machine hôte :

```
docker network ls
```

Vous devez certainement avoir déjà les réseaux **host**, **bridge** et **none**. Ils ne doivent pas être supprimés, ce sont des réseaux spéciaux de Docker.

Pour supprimer un réseau :

```
docker network rm nom_du_reseau
```

Supprimez votre réseau **mon_rezo**.

La suppression d'un réseau n'est possible que si aucun conteneur n'y est branché.

Exercice 5 - Attachement à un réseau

Attachement

Quand on branche un réseau à un conteneur on dit qu'on attache le conteneur au réseau.

Un conteneur peut être attaché à zéro (cas par défaut), un ou plusieurs réseaux.

A contrario du mappage de port qui ne peut se faire qu'à la création d'un conteneur, l'attachement à un réseau peut se faire à la création du conteneur (pas à la création du réseau) ou a posteriori sur un conteneur actif.

On ne peut pas attacher, à un réseau, un conteneur qui est arrêté.

Quand un conteneur s'arrête, il est automatiquement détaché de tous ses réseaux.

À la création du conteneur

Voici la syntaxe de l'option à ajouter pour attacher un conteneur à un réseau au moment de sa création :

```
docker container run --network nom_du_réseau ...
```

Rappel : le réseau **nom_du_réseau** doit exister au préalable.

Sur un conteneur actif

Voici la syntaxe à utiliser pour attacher un réseau existant à un conteneur actif :

```
docker network connect nom_du_réseau id_ou_nom_du_conteneur
```

Détachement

L'action de débrancher un réseau d'un conteneur s'appelle détacher le conteneur du réseau. C'est une action qui est faite automatiquement à l'arrêt du conteneur mais qui peut aussi être faite sur un conteneur actif sans aucun risque. Il perd évidemment toute connexion active sur ce réseau au moment du détachement.

Voici la syntaxe à utiliser pour détacher un réseau d'un conteneur actif :

```
docker network disconnect nom_du_réseau id_ou_nom_du_conteneur
```

DNS

Ce qui est pratique avec Docker et ses réseaux est qu'il entretient une petite DNS locale et que les conteneurs peuvent s'adresser aux autres conteneurs sur le réseau privé grâce à leur petit nom de conteneur !

Expérimentons

Notez que **ping** est une commande très pratique qui permet de voir si une machine donnée est présente sur le réseau. Si elle est présente, elle répondra au **ping**. C'est une commande standard de Linux, elle n'est pas particulière à Docker.

Voici les étapes de l'expérimentation :

1) Dans un terminal qu'on nommera ici **T1**, créez un conteneur nommé **pipo** (passez l'option **--name pipo** à la commande **docker container run**) sur la base d'une image **r408-ping**, en mode interactif.
2) Dans un terminal qu'on nommera ici **T2**, créez un conteneur nommé **lulu** sur la base d'une image **r408-ping**, en mode interactif.
3) Dans **T1**, tapez un **ping lulu**. En principe il ne doit pas connaître **lulu** !

4) Dans un terminal qu'on nommera **T3**, créez un réseau nommé **test**
5) Dans **T3**, attachez les deux conteneurs à ce réseau **test**.
6) Dans **T1**, refaites un **ping lulu**. Ça doit être mieux et il doit obtenir une réponse de **lulu** à son **ping** !!!
7) Dans **T1**, le **ping** doit rester tourner sans fin, laissez-le faire
8) Dans **T3**, détachez l'un des deux conteneur du réseau **test**
9) Que s'est-il passé dans **T1** ?

Exercice 6 - Watch dog

Un Watch dog en informatique est un processus qui surveille quelque chose (il s'agit souvent de l'état d'un service) et qui fait une action donnée si l'état change.

Vous allez écrire, en **PHP**, un Watch dog qui va surveiller qu'un conteneur **nginx** est bien actif et en état de fonctionnement.

Pour ce faire vous allez créer :

- Un réseau nommé **watchdog** sur lequel vous attacherez les deux conteneurs suivants.
- Un conteneur **nginx** (reprenez celui de l'Exercice 3 avec votre mini site Web). Cette fois-ci, prévoyez de lui donner un nom au lancement, ce sera plus pratique.
- Un conteneur **watchdog** sur la base d'une image **r408-php** qui va exécuter un script PHP faisant ceci dans une boucle infinie :
    - Récupérer la page d'accueil de votre mini site
    - En cas d'erreur, afficher un message
    - Faire une pause de 5 secondes
    - Boucler

  L'image **r408-php** peut exécuter un script en passant au **docker container run** les arguments suivants : **php chemin_vers_script.php**

Testez en arrêtant et en redémarrant le conteneur **nginx**.

Exercice 7 - Image watchdog

Créez un **Dockerfile** permettant de construire une image **watchdog:1.0** qui doit contenir le nécessaire pour lancer un watchdog sur une URL passée en paramètre (à la suite du nom de l'image lors du **docker run**).

Rappel : la commande lancée dans un conteneur est construite par la concaténation des valeurs de **ENTRYPOINT** + **CMD**. Adaptez donc vos **ENTRYPOINT** et **CMD** pour que ça puisse lancer votre script avec l'URL passée en paramètre du **docker run**.

Testez avec deux conteneurs Web différents (ayant des noms différents) et vérifiez que votre image est bien capable de gérer ces deux situations à l'aide de l'URL de paramètre.

Exercice 8 - curl

La commande **`curl`** permet de récupérer n'importe quel contenu sur le Web.

Comme toute commande, elle retourne un code **`0`** si tout s'est bien passé et une autre valeur en cas d'erreur.

Rappel : en Bash, la variable **`$?`** contient le code de retour de la dernière commande lancée.

Créez un script Bash qui fait la même chose que le script PHP et testez-la dans un conteneur créé depuis une image **`r408-ubuntu:23.04`**.

Créez un **`Dockerfile`** permettant de construire une image **`watchdog-bash:1.0`** qui doit faire la même chose que **`watchdog:1.0`**. Vous utiliserez une image **`r408-ubuntu:23.04`** comme image de base.

Comparez les tailles de vos images **`watchdog:1.0`** et **`watchdog-bash:1.0`**