TP MDR : installation et exploitation d'un cluster MongoDB sur Docker, embarqué sur Raspberry

Avant propos : *Les conditions de réalisation de ce TP sont loin d'être idéales : manque de matériel, difficultés de configuration, infrastructure réseau inadaptée … Mais celles du monde de l'entreprise ne sont pas toujours mieux. Au travers de ce TP, nous allons, au delà de l'objectif principal, rencontrer des soucis et proposer de nombreuses astuces pour contourner les difficultés que nous allons rencontrer. En ce sens, la portée pédagogique de ce TP mettra également l'accent sur la résolution de problèmes en situation réelle, qui ne seraient pas rencontrées dans des conditions idéales de travail.*

Ce TP se divise en deux parties presque indépendantes :

1 Installation du Rasberry Pi et déploiement des outils de travail
2 Déploiement d'un cluster mongoDB en containers virtuels via Docker

1 Installation et exploitation du Raspberry Pi

Dans un monde idéal (avec des moyens, du budget, un responsable des achats qui répond aux demandes de devis matériel …) , nous aurions un écran, un clavier, une souris et un cable hdmi, qu'il suffirait de connecter au raspberry pour l'utiliser comme un vrai ordinateur et procéder à une installation facile et rapide. Or nous sommes à l'ESIREM. Nous allons donc devoir nous débrouiller avec un minimum de matériel et un peu (beaucoup) de chance. Et d'astuce. L'objectif de cette section est de **mettre en place un système opérationnel sur Raspberry, connecté au réseau, et pleinement contrôlable par l'utilisateur**. Cet objectif est rendu particulièrement difficile par l'absence d'écran et de clavier, qui nous empêche de communiquer directement avec le matériel.

> Avec un clavier, une souris et un écran, il suffirait d'utiliser le raspberry Pi comme un ordinateur complet, d'insérer la carte microSD contenant la distribution NOOBS qui permet l'installation du système d'exploitation, et de suivre les étapes affichées à l'écran en validant successivement
>
> *Et voilà !*

L'approche proposée dans l'encadré précédent requiert du matériel que nous n'avons pas. Nous avons seulement un lecteur de carte microSD. Nous allons donc envisager le Raspberry Pi comme un système embarqué sans clavier ni écran. La réalisation de cet objectif va donc suivre le cheminement suivant, qui se découpe en deux étapes :

1. Installation en amont du système d'exploitation **archLinux** sur une carte microSD connectée à l'ordinateur, que nous insérerons ensuite dans le raspberry Pi
2. Déploiement physique du matériel : allumage et connexion physique au réseau, et accès au raspberry Pi via un shell ssh pour configuration des éléments du réseau

Par chance, le service `ssh-server` est configuré et tourne par défaut sur archLinux. Par malchance, l'accès à ce serveur risque d'être difficile si l'on ne dispose pas d'un serveur dhcp pour configurer automatiquement le raspberry Pi sur le réseau. Nous verrons comment contourner cette difficulté.

1.1 Installation du système d'exploitation

1.1.1 Récupération du système d'exploitation archlinux

Nous effectuerons tout en ligne de commande en tant qu'utilisateur `root`, pour disposer des droits administrateurs.

Attention : il y a une différence entre être utilisateur root d'une part, et exécuter des commandes avec les droits administrateurs au moyen de la commande sudo d'autre part : L'environnement root est différent de l'environnement d'un utilisateur standard + droits administrateurs. Par exemple le répertoire de travail de départ n'est pas le même. Pour éviter toute confusion et compte tenu de la nécessité d'exécuter certains scripts en utilisateur root exclusivement, la solution de facilité ici sera de continuer en étant utilisateur root exclusivement.

Dans la suite, les encadrés en gris clair correspondent aux commandes à taper dans le terminal de notre ordinateur, exception faite du prompt # ou `$`. Les éléments sur fond gris foncé correspondent aux résultats de l'affichage. Nous passons tout d'abord en mode administrateur avec la commande suivante (le mot de passe utilisateur sera demandé) :

```
$ sudo su
```

L'invite de commande doit avoir changé et se terminer par un # plutôt qu'un `$`. Nous créons un répertoire `raspberryInstall` à la racine de du répertoire de travail de l'utilisateur `root` et travaillerons par la suite à partir de ce dernier exclusivement :

```
# mkdir ~/raspberryInstall
# cd ~/rasbberryInstall/
```

Nous téléchargeons ensuite la version de ArchLinux que nous souhaitons utiliser. Il existe trois versions principales :

- pour Raspberry 1, qui disposent d'une d'un processeur ARMv6
- pour Raspberry 2/3, qui disposent d'une d'un processeur ARMv7
- pour Raspberry 3, uniquement qui disposent d'une d'un processeur ARMv7

Ces différentes versions sont téléchargeables aux adresses suivantes :

- `http://os.archlinuxarm.org/os/ArchLinuxARM-rpi-latest.tar.gz`
- `http://os.archlinuxarm.org/os/ArchLinuxARM-rpi-2-latest.tar.gz`
- `http://os.archlinuxarm.org/os/ArchLinuxARM-rpi-3-latest.tar.gz`

Pour des soucis de compatibilité, nous prendrons la version pour Rasberry 2/3. Nous utilisons `wget` pour récupérer la version correspondante :

```
# wget http://os.archlinuxarm.org/os/ArchLinuxARM-rpi-2-latest.tar.gz
```

Le résultat de cette commande est le téléchargement dans le répertoire courant de l'archive `ArchLinuxARM-rpi-2-latest.tar.gz` . que nous utiliserons par la suite

1.1.2 Préparation de la carte microSD :

Nous allons formater la carte microSD pour qu'elle puisse contenir par la suite le système archLinux, puis copier le système d'exploitation dessus.

a) Identification du nom de la carte microSD

Il nous faut d'abord déterminer le nom du périphérique correspondant à la carte.

Sous linux, nous utilisons la commande `fdisk` avec l'option `–l` (list) avant et après insertion de la carte microSD sur l'ordinateur. Tapez la commande suivante avant de connecter la carte microSD à l'ordinateur

```
# fdisk -l
```

Puis insérez la carte dans l'ordinateur et retaper la commande. Analysez les nouvelles lignes pour déterminer le nom du périphérique correspondant à la carte.Par exemple, si les lignes suivantes sont apparues :

```
Disque /dev/sdc : 15 GiB, 16088301568 octets, 31422464 secteurs
Unités : sectors of 1 * 512 = 512 octets
Sector size (logical/physical): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x000afe9d
```

Cela signifie que notre carte microSD possède le nom `/dev/sdc` Si des lignes similaires sont également apparues :

```
Périphérique Amorçage   Start      Fin Secteurs  Size Id Type
/dev/sdc1                8192  2289062  2280871  1,1G  e W95 FAT16
(LBA)
/dev/sdc2             2289063 31422463 29133401 13,9G  5 Étendue
/dev/sdc5             2293760  2359293    65534   32M 83 Linux
/dev/sdc6             2359296  2488319   129024   63M  c W95 FAT32
(LBA)
/dev/sdc7             2490368 31422463 28932096 13,8G 83 Linux
```

Cela signifie que la carte microSD n'est pas vierge, et contient déjà plusieurs partitions, respectivement nommées `/dev/sdc1`, `/dev/sdc2`, `/dev/sdc5`, `/dev/sdc6`, et `/dev/sdc7` qu'il faut supprimer avant.

Pour éviter toute confusion par la suite, nous appellerons `/dev/sdc` le nom de la carte microSD, à adapter en fonction de votre configuration. Attention, ce nom est susceptible de changer si vous débranchez puis rebranchez la carte.

b) Suppression du contenu de la carte et création d'une nouvelle table de partition

Nous allons utiliser la commande `fdisk` pour supprimer toutes les partitions existantes de la carte microSD, et créer une nouvelle table de partition.

Lancer la commande fdisk avec le nom de la carte en paramètres :

```
fdisk  /dev/sdc
```

L'utilisation de la commande `fdisk` est minimale. Les sous-commandes sont accessibles au travers d'options à indiquer via la lettre correspondante. Il vous est par exemple possible de :

- lister les options disponibles : taper `m`
- lister la table des partitions courantes : taper `p`
- supprimer une partition particulière: taper `d` puis le numéro de la partition à supprimer
- supprimer toutes les partitions d'un coup : taper `o`
- enregistrer puis quitter l'utilitaire : taper `w`
- quitter sans enregistrer : taper `q`

Tapez `o` pour supprimer tout le contenu, puis `p` pour vérifier que la table des partitions est bien vide.

c) Création et formatage des nouvelles partitions de la carte

Pour que la carte microSD puisse fonctionner sur le rasbperry, il va nous falloir créer deux partitions particulières :

1) Une partition primaire de `boot` de 100 Mo, qui sera formatée avec un système de fichiers FAT
2) Une seconde partition racine (`root`), formatée avec un système de fichier ext4 qui contiendra la distribution archLinux

Lancer `fdisk` avec le nom de la carte microSD, par exemple :

```
fdisk  /dev/sdc
```

Nous créons d'abord la première partition :

- Choisir l'option `n` pour créer une nouvelle partition
- Puis taper `1` pour indiquer qu'il s'agit de la première partition de la carte. Valider pour que cette partition se situe sur le premier secteur par défaut de la carte.
- Taper `+100M` pour le dernier secteur.
- Ensuite taper `t` pour changer le type de partition, et choisir `c` pour que la première partition soit de type W95 FAT32 (LBA).

Voici un exemple similaire à ce qui devrait avoir été affiché

```
Commande (m pour l'aide) : n
```

```
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Numéro de partition (1-4, 1 par défaut) : 1
Premier secteur (2048-31422463, 2048 par défaut) :
Last sector, +sectors or +size{K,M,G,T,P} (2048-31422463, 31422463 par
défaut) : +100M

Created a new partition 1 of type 'Linux' and of size 100 MiB.

Commande (m pour l'aide) : t
Selected partition 1
Partition type (type L to list all types): c
Changed type of partition 'Linux' to 'W95 FAT32 (LBA)'.
```

Nous créons ensuite la seconde partition de manière similaire, en adaptant les valeurs aux exigences de la partition :

- Choisir l'option n pour créer une nouvelle partition
- puis 2 pour indiquer qu'il s'agit de la première partition de la carte
- Valider deux fois sans changer les valeurs de début de secteur et fin de secteur.

Voici un exemple similaire à ce qui devrait avoir été affiché :

```
Commande (m pour l'aide) : n
Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
Select (default p): p
Numéro de partition (2-4, 2 par défaut) : 2
Premier secteur (206848-31422463, 206848 par défaut) :
Last sector, +sectors or +size{K,M,G,T,P} (206848-31422463, 31422463
par défaut) :

Created a new partition 2 of type 'Linux' and of size 14,9 GiB.
```

Nous vérifions avec la commande p que les deux partitions sont bien présentes. L'affichage devrait contenir des lignes similaires :

```
Périphérique Amorçage  Start      Fin Secteurs  Size Id Type
/dev/sdc1              2048   206847   204800  100M  c W95 FAT32 (LBA)
/dev/sdc2            206848 31422463 31215616 14,9G 83 Linux
```

Notons que la première partition s'appelle dans l'exemple `/dev/sdc1`, et la seconde partition `/dev/sdc2`. Puis nous enregistrons et quittons la commande `fdisk` avec la commande `w`.

Nous formatons ensuite les nouvelles partitions aux systèmes de fichiers correspondants avec les commandes `mkfs.vfat` et `mkfs.ext4` que nous lançons en précisant à chaque fois la partition correspondante :

```
mkfs.vfat /dev/sdc1
mkfs.ext4 /dev/sdX2
```

Notre clé est prête à accueillir le système d'exploitation ArchLinux

1.1.3 Installation de ArchLinux sur la carte microSD

L'archive archLinux, contient les fichiers et répertoires d'un système d'exploitation propre à Linux, ainsi qu'un répertoire particulier boot. Nous allons placer le contenu du répertoire boot sur la partition boot (dans l'exemple `/dev/sdc1`) de la carte, et le reste sur la seconde partition (dans l'exemple `/dev/sdc2`). Pour cela nous créons deux répertoires `SDboot` et `SDroot` qui seront les points de montage pour les deux partitions :

```
mkdir SDboot
mkdir SDroot
```

puis on monte les deux partitions de la carte dans ces répertoires :

```
mount /dev/sdc1 SDboot
mount /dev/sdc2 SDroot
```

Désormais, ce qui est copié dans le répertoire `SDboot` le sera en réalité sur la partition `/dev/sdc1`, et respectivement dans `SDroot` pour `/dev/sdc2`.

Ensuite on décompresse tout le contenu de l'archive sur la partition `/dev/sdc2`, en utilisant la commande `tar`, couplée aux options `xzvf`, et en précisant comme répertoire de destination `SDroot` grâce à l'option `–C`.

```
tar xzvf ArchLinuxARM-rpi-2-latest.tar.gz –C SDroot
```

Enfin on déplace le contenu du répertoire `boot` de `/dev/sdc2` vers la partition de `/dev/sdc1` :

```
mv SDroot/boot/* SDboot
```

Nous démontons enfin les partitions montées pour éjecter correctement la carte :

```
umount SDroot
umount SDboot
```

La carte est prête et peut être insérée dans le Raspberry Pi.

1.2 Déploiement physique du matériel : allumage et connexion physique au réseau

Nous allons démarrer le raspberry sur le système d'exploitation ArchLinux. Insérez la carte microSD dans le lecteur du raspberry, puis branchez le rasberry sur l'alimentaiton secteur. Une diode rouge témoigne de la mise sous tension. Une diode verte clignotant par intermittences indique que le raspberry est en cours de démarrage. Au bout d'une minute environ, le système est prêt

Une étape importante consiste à connecter le Raspberry à un réseau, pour bénéficier ensuite d'un accès `ssh` et terminer la configuration du matériel. Dans un monde idéal, nous disposerions d'un switch connecté à un réseau filaire sur lequel est présent un serveur dhcp, et un routeur gateway. Il suffirait alors simplement de brancher un cable réseau rj45 entre le raspberry et le switch, laisser le serveur dhcp attribuer une adresse au raspberry et configurer la passerelle vers l'extérieur (internet), déterminer l'adresse IP du raspberry et se connecter dessus.

Or nous sommes à l'ESIREM. Nous allons donc nous connecter au Raspberry sans switch, ni serveur dhcp. Cette configuration nécessite néanmoins que notre station de travail dispose d'une interface ethernet non utilisée. Naturellement, avec un peu de matériel, nous aurions pu facilement simplifier cette partie.

1.2.1 Accès à un shell ssh sur rasberry

Pour accéder au shell `ssh` du rasberry, il va nous falloir dans un premier temps mettre sur le même réseau notre station de travail et notre Raspberry. Sans switch, nous connectons directement le raspberry Pi à l'interface réseau disponible notre station de travail. Nous supposerons qu'il s'agit de l'interface eth1, nom d'interface qu'il conviendra d'adapter par la suite. Et après ? Il nous manque toujours un adressage réseau pour espérer communiquer …

Du coté de la station de travail, affichons l'adresse IP de la carte réseau connectée au raspberry avec la commande `ifconfig` :

```
ifconfig eth1

eth1      Link encap:Ethernet  HWaddr 08:00:27:2d:e1:6e
          inet adr:169.254.108.53 Bcast: 169.254.255.255
Masque:255.255.0.0
          adr inet6: fe80::184f:5ed1:231f:aa87/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

```
          Packets reçus:6 erreurs:0 :0 overruns:0 frame:0
          TX packets:61 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          Octets reçus:1175 (1.1 KB) Octets transmis:6939 (6.9 KB)
```

La bonne nouvelle, c'est que notre système a configuré automatiquement une adresse IP, ici `169.254.108.53`, avec un masque réseau de `255.255.0.0`. Cette configuration est la conséquence de l'application d'auto-ip en l'absence de serveur DHCP et de configuration statique sur la carte.

Il suffirait que le rasberry Pi dispose d'un mécanisme similaire pour se voir attribuer automatiquement une adresse ip sur le réseau `169.254.0.0/16`. En pareille circonstance, il ne resterait « plus qu'à » trouver l'adresse ip du raspberry Pi parmi la plage `169.254.0.1 – 169.254.255.254`, soit :

- en pingant l'adresse broadcast (mais certains hôtes sont configurés pour ne pas répondre aux ping broadcast)
- en pingant toutes les adresses IP (un logiciel comme `nmap` ferait l'affaire),
- soit en sniffant les paquets sur le réseau avec `wireshark` ou `tcpdump` par exemple.

Or nous n'avons pas de chance : ArchLinux ne dispose pas d'une telle configuration par défaut. En revanche, nous remarquons également qu'une adresse en IPv6 a été attribuée à notre carte : ici `fe80::184f:5ed1:231f:aa87`. Il s'agit d'une adresse lien-local. Les spécificités des adresses IPv6 feront l'objet d'un cours à part entière. Il suffit que le Raspberry dispose lui aussi d'une adresse lien-local en IPv6 pour que nous puissions nous connecter dessus. Et il s'avère que c'est le cas !

Déterminons l'adresse IPv6 du Raspberry : nous effectuons un ping broadcast IPv6 sur l'interface `eth1`, grace à la commande `ping6`, en utilisant une adresse IPv6 spéciale : `ff02::1`.

```
ping6 –I eth1 ff02::1
```

Tous les hôtes directement connectés et ayant une adresse IPv6 répondent. Regardons les résultats :

```
ping6 -I enp0s3  ff02::1

PING ff02::1(ff02::1) from fe80::e18c:5940:d9a2:73ce enp0s3: 56 data
bytes
64 bytes from fe80::e18c:5940:d9a2:73ce: icmp_seq=1 ttl=64 time=0.028
ms
64 bytes from fe80::ba27:ebff:fe29:4b9c: icmp_seq=1 ttl=64 time=0.049
ms
```

bingo ! l'adresse `fe80::ba27:ebff:fe29:4b9c` est apparue! Il s'agit de l'adresse IPv6 auto-attribuée du Raspberry. Le client `ssh` de notre station de travail supporte nativement IPv6. Le login / mot de passe par défaut du rasbperry est `alarm` / `alarm`. Il faudra rajouter à l'adresse du Raspberry '`%eth1`' pour préciser sur quelle interface elle est attribuée. Ainsi la commande suivante nous permet de nous connecter au Raspberry :

```
ssh fe80::ba27:ebff:fe29:4b9c%eth1 –l alarm
```

Nous obtenons l'affichage suivant :

```
alarm@fe80::ba27:ebff:fe29:4b9c%eth1's password:
Welcome to Arch Linux ARM

     Website: http://archlinuxarm.org
       Forum: http://archlinuxarm.org/forum
         IRC: #archlinux-arm on irc.Freenode.net
[alarm@alarmpi ~]$
```

Nous avons désormais un shell ssh sur le Rasberry, grâce à une connexion IPv6 un peu fortuite. L'étape suivante consiste à mettre en place une connexion IPv4 stable et efficace : nous allons rajouter un adressage IPv4 dans un premier temps, puis lui donner accès à internet.

Dans la suite, pour augmenter la lisibilité de ce TP, nous utiliserons un style légèrement différent pour désigner les commandes à taper sur le shell du rasberry PI, par rapport à celles à taper sur la station de travail.

1.2.2 Connexion du raspberry à internet

Pour la suite de l'installation, nous avons besoin que le raspberry puisse se connecter à internet.

Le système de droits administrateur est différent sur archLinux par rapport à ubuntu. Il n'y a pas de commande sudo. Il existe nénamoins un compte administrateur dont le login / mot de passe est `root` / `root`. L'utilisation de ce compte est nécessaire pour modifier la configuration du Raspberry. On se loggue d'abord sur ce compte avec la commande `su root` :

```
[alarm@alarmpi ~]$ su root
Password:
[root@alarmpi alarm]#
```

Si un réseau est disponible en wifi, il nous suffit de configurer le Raspberry pour qu'il se connecte à ce dernier simplement avec la commande suivante et de se laisser guider :

```
wifi-menu
```

Attention, un réseau avec une authentification ouverte mais un portail captif est généralement déconseillée car plus difficile à configurer sans interface graphique.

Nous supposerons que nous n'avons pas de réseau sur lequel connecter le Raspberry. Nous allons donc utiliser notre station de travail comme passerelle, de manière similaire à ce qui a été fait en TP de réseau en 3eme année.

Nous utiliserons le réseau `192.168.1.0/24`. Nous attribuons l'adresse IP `192.168.5.1` à la station de travail qui fera office de routeur passerelle, soit via l'interface graphique, soit directement en ligne de commande avec la commande suivante :

```
ifconfig eth1 192.1685.1/24
```

Notons que cette commande est temporaire, et qu'un redémarrage de la station de travail entrainera la perte de cette IP. Il faudra soit la remettre, soit sauvegarder la configuration.

Nous attribuons l'adresse IP 192.168.5.2 au Raspberry. Regardons le nom de l'interface ethernet du raspberry :

```
[root@alarmpi alarm]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.5.2  netmask 255.255.255.0  broadcast
192.168.5.255
        inet6 fe80::ba27:ebff:fe29:4b9c  prefixlen 64  scopeid
0x20<link>
        ether b8:27:eb:29:4b:9c  txqueuelen 1000  (Ethernet)
        RX packets 10180  bytes 2087878 (1.9 MiB)
        RX errors 0  dropped 2292  overruns 0  frame 0
        TX packets 3347  bytes 775838 (757.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Ici l'interface du Rasberry est `eth0`. Gardons en tête que dans cet énoncé, `eth0` désigne la carte réseau du Raspberry, et `eth1` celle de la station de travail, et que `eth0` et `eth1` sont reliés directement par un câble ethernet. La commande suivante suffit à attribuer l'adresse IP `192.168.5.2` sur l'interface `eth0` du Raspberry :

```
ifconfig eth1 192.1685.2/24
```

A ce point précis, nous pouvons éventuellement couper la connexion ssh en IPv6 et tester la connexion ssh en IPv4 pour s'assurer que tout fonctionne bien :

```
ssh 192.168.5.1 –l alarm
```

On revient ainsi sur le shell du Raspberry. On se relogue en tant qu'utilisateur `root`, (commande mot de passe `root`),

```
su root
```

Nous continuons la configuration pour ajouter un accès vers Internet : on ajoute une passerelle par défaut sur le Raspberry :

```
route add default gw 192.168.5.1
```

On vérifie l'exactitude de la table de routage avec la commande `route -n`:

```
[root@alarmpi ~]# route -n
Kernel IP routing table
Destination  Gateway       Genmask         Flags Metric Ref    Use Iface
0.0.0.0     192.168.5.1    0.0.0.0          UG    0      0        0 eth0
192.168.5.  0 0.0.0.0      255.255.255.0    U     0      0        0 eth0
```

Pour vérifier la configuration, nous pingons la machine 8.8.8.8 (qui est un serveur de google)…

```
[root@alarmpi ~]# ping 8.8.8.8
```

et ca ne marche pas ! Ce qui est normal, ou pas. Si le message d'erreur affiché est :

```
connect: Network is unreachable
```

c'est qu'il y a eu un problème de configuration de la route par défaut. En revanche si nous avons un affichage comme suit, avec un taux de perte de 100% après 10 secondes :

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
^C
--- 8.8.8.8 ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9399ms
```

Aucun soucis, tout est normal. Il nous faut « simplement » activer le routage sur la station de travail, ainsi que la translation d'adresse. On tape donc sur la station de travail les commandes suivantes (voir TP réseau 3A pour le détail de ces commandes) :

On active le routage des paquets IPv4 sur la station de travail :

```
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
```

On active ensuite la fonction de translation d'adresse NAT, en remplaçant dans la ligne suivante ethX par le nom de l'interface de la station de travail qui est connectée à Internet. Attention, cette interface n'est PAS eth1 dans notre exemple (eth1 étant connectée au Raspberry).

```
iptables -t nat -A POSTROUTING -s 192.168.5.0/24 -o ethX -j MASQUERADE
```

Désormais, la connexion internet doit être disponible sur le raspberry. Nous le vérifions via la commande suivante pour s'assurer du routage IP

```
ping 8.8.8.8

PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=56 time=8.69 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=56 time=8.45 ms
^C
--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 8.453/8.574/8.695/0.121 ms
```

L'étape suivante ne devrait normalement pas avoir à être réalisée, car la configuration DNS est automatiquement définie pour utiliser les serveurs de Google. Mais l'université filtrant ces serveurs, nous allons devoir les ajouter. Dans un système usuel, il faudrait simplement les ajouter dans le fichier **/etc/resolv.conf** . Mais sur la distribution archlinux, il nous faut plutot éditer le fichier

`/etc/systemd/resolved.conf` qui correspond au fichier de configuration du service local s'occupant de la résolution des DNS. Au sein de ce fichier, on ajoute nos DNS en remplacant la ligne contenant FallbackDNS par la ligne suivante :

nano `/etc/systemd/resolved.conf`

#chercher et modifier la ligne FalbackDNS en :
`FallbackDNS=193.50.50.2 193.50.50.6 2001:4860:4860::8888 2001:4860:4860::8844`
où `193.50.50.2` et `193.50.50.6` sont les serveurs DNS de l'université de Bourgogne. Nous enregistrons (ctrl + o) et nous quittons (ctrl +x).

Pour etre sûr que la modification a été prise en compte, on arrete et on relance le service systemd :

```
[root@alarmpi ~]# ps -aux | grep reso
```

on note le pid et on lui envoie la commande kill. Le système se relancera automatiquement.

Nous vérifions enfin que la résolution des noms de domaine fonctionne :

```
ping yahoo.fr

PING yahoo.fr (98.137.236.24) 56(84) bytes of data.
64 bytes from aviate.yahoo.com (98.137.236.24): icmp_seq=1 ttl=45
time=183 ms
64 bytes from aviate.yahoo.com (98.137.236.24): icmp_seq=2 ttl=45
time=181 ms
^C
--- yahoo.fr ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 181.103/182.266/183.429/1.163 ms
[root@alarmpi alarm]#
```

Au terme de ces 8 pages d'énoncé, nous disposons *enfin* d'un OS sur Rasberry opérationnel et sur lequel nous allons pouvoir déployer notre système.

Mais la configuration ne sera pas sauvegardée après redémarrage : il faudra rétablir l'accès comme précédemment.

2 Mise en œuvre d'une architecture MongoDB type *replica set*

Les choses commencent à se compliquer dans cette partie. Nous allons mettre en place un cluster de de base de donnée. Afin de limiter dans un premier temps la complexité de l'installation, nous allons partir sur une architecture simple comprenant un « ***replica set*** », c'est à dire un groupe de processus répartis mongoDB qui maintiennent le ***même ensemble de données***. Un avantage des Replica set est de fournir une haute redondance et une haute disponibilité. Il s'agit de la configuration de base pour les déploiements en production. . Le Replica set est composé des éléments suivants :

- Des nœuds de données (data bearing nodes), sur lesquels sont stockés les données.
- Parmi les nœuds de données, un ***nœud primaire***. Ce nœud unique reçoit les opérations d'écriture, et il est le seul à pouvoir les confirmer. Les autres nœuds sont les nœuds
- Un nœud arbitre, qui intervient dans le processus d'élection d'un nouveau nœud primaire en cas de panne. Ce nœud est optionnel et ne devrait pas disposer d'une puissance de calcul élevée. Il est surtout utile si le nombre de nœuds est pair. Nous ne l'utiliserons pas ici.

2.1 Architecture à déployer

Nous allons déployer trois nœuds mongoDB via l'interface de virtualisation légère docker. Le premier nœud déployé sera appelé `manager`, et permettra de communiquer la configuration aux autres nœuds, appelés respectivement `worker1` et `worker2`. Chacun de ces nœuds tournera dans un container docker, qui tournera lui-même dans une machine virtuelle docker afin d'obtenir une vision logique réseau conforme avec un réseau physique.

Néanmoins, nous veillerons à ce que l'espace de stockage, même s'il ne sera pas partagé entre les containers, soit persistant. Pour cela, nous créerons un lien entre un répertoire de la machine virtuelle et le répertoire censé contenir la configuration et les données de la base sur chaque nœud mongo.

2.2 Mise en œuvre de l'architecture

2.2.1 Installation des outils sur ArchLinux sur le raspberry

ArchLinux utilise le gestionnaire de package `pacman`. Tout d'abord nous mettons à jour les packages et la liste des packages disponibles du Rasberry afin d'éviter tout problème de dépendance avec la commande suivante :

```
$ pacman –Syu
```

Ensuite nous installons: `docker`

```
$ pacman –S docker
```

Il va nous falloir redémarrer le Raspberry car le noyau a pu être changé … et reconfigurer les adresses IP.

On lance le daemon docker :

```
systemctl start docker.service
```

C'est parti pour un remake d'Inception dans le monde formidable de la virtualisation.

2.2.2 Création des volumes de stockage pour chaque container, et autres prérequis

Nous allons créer tout d'abord trois containeurs virtuels nommées manager, worker1, worker2. Chaque containeur va agir comme un propre site indépendant, et disposera de son propre espace de stockage, virtualisé en tant que volume, que l'on peut voir simplement comme un répertoire. Il sera cependant possible d'observer facilement depuis le raspberry le contenu de chaque volume. Nous créeons d'abord les trois volumes pour chaque containeur avec la commande `docker volume` :

```
$ docker volume create --name mongo_storage_manager
$ docker volume create --name mongo_storage_worker1
$ docker volume create --name mongo_storage_worker2
```

On peut utiliser les sous commandes `ls` et `inspect` pour lister l'ensemble des volumes, ou obtenir des informations complémentaires, comme le chemin physique d'un volume. Par exemple listons l'ensemble des volumes docker :

```
$ docker volume ls
DRIVER              VOLUME NAME
local               mongo_storage_manager
local               mongo_storage_worker1
local               mongo_storage_worker2
```

Nous récupérons l'image du container mongo pour la déployer par la suite.

```
2.2.3 $ docker pull nonoroazoro/rpi-mongodb
```

Enfin, nous créons un réseau docker, ici 172.16.0.0/16, nommé monClusterMongo. Ce réseau simulé permettra aux différentes instances du cluster de communiquer comme si elles se trouvaient sur un vrai réseau physique.

```
$ docker network create --subnet=172.16.0.0/16 monClusterMongo
```

Création et lancement sécurisé du nœud principal manager

La création du nœud principal va se faire en plusieurs phases :

1. Nous allons d'abord créer le noeud manager en lançant le container applicatif sans authentification
2. puis nous placerons les éléments d'authentification dans le volume de ce container
3. Puis nous allons supprimer (!) le containeur applicatif, et le recréer en le lancant cette fois avec les éléments d'authentification.

Supprimer et relancer le container peut paraître bizarre, mais pas tant que ca finalement ! n'oublions pas que comme les éléments de configuration sont persistants car stockés sur un volume de stockage, il nous suffit de relancer le container applicatif en utilisant le volume précédemment créé. Ici, le fait de supprimer le containeur applicatif puis de le relancer avec authentification revient simplement à stopper un processus et à le relancer. Sauf qu'on va pouvoir le relancer en passant les éléments de configuration directement depuis docker

Nous créons le containeur virtuel, sur le réseau monClusterMongo, sans configuration, de la facon la plus simple possible. Voyez déjà l'ajout de la commande --ip qui permet de spécifier l'adresse IP à utiliser sur ce container. Ici l'adresse de 172.16.0.100 correspond bien à notre architecture, et se situe bien sur le réseau monClusterMongo. Notez aussi que pour des raisons de facilité, on mappe le port 27017 de notre raspberry sur le port 27017 de notre container pour pouvoir y accéder plus rapidement via un client mongo.

```
$ docker create --name manager -v mongo_storage_manager:/data --net
monClusterMongo --ip 172.16.0.100 -p 27017:27017  nonoroazoro/rpi-mongo
```

Puis nous lançons le container

```
$ docker start  manager
```

Nous allons ensuite configurer manager et le considérer comme faisant partie d'un cluster de nœuds mongodb. Pour faire partie du même cluster, les nœuds doivent partager une même clé privée. Nous générons tout d'abord cette clé dans un fichier mongo-keyfile, stockée en local sur notre raspberry.

```
$ openssl rand -base64 741 > mongo-keyfile
```

Sur le container applicatif `manager`, nous créons ensuite un sous-répertoire `/data/keyfile/` qui va contenir la clé.

```
$ docker exec manager bash -c 'mkdir /data/keyfile'
```

Notons que comme ce sous-répertoire est dans le répertoire /data, il sera conservé dans le volume `mongo_storage_manager`, et sera persistant même si le conteneur applicatif manager venait à être détruit puis recréé. Nous copions ensuite la clé dans ce répertoire (avec la commande docker cp).

```
$ docker cp mongo-keyfile manager:/data/keyfile/
```

nous changeons enfin les droits et le propriétaire du répertoire /data en mongodb, pour que notre container puisse avoir accès sans problème à ces fichiers (si les permissions sont trop ouvertes, le serveur ne se lancera pas par la suite).

```
$ docker exec manager bash -c 'chown -R mongodb:mongodb /data'
$ docker exec manager bash -c 'chmod 600 /data/keyfile/mongo-keyfile'
```

Nous pouvons vérifier que tout s'est bien passé en listant le contenu de /data et /data/keyfile. Le résultat du listing de ces deux répertoires doit être (à peu près) le suivant :

```
$ docker exec manager bash -c 'ls  -l /data/'
total 16
total 12
drwxr-xr-x 2 mongodb mongodb 4096 Mar 21 10:18 configdb
drwxr-xr-x 3 mongodb mongodb 4096 Mar 21 10:19 db
drwxr-xr-x 2 mongodb mongodb 4096 Mar 21 10:19 keyfile
```

```
$ docker exec manager bash -c 'ls  -l /data/keyfile/'
```

```
total 4
-rw------- 1 mongodb mongodb 1004 Mar 21 09:34 mongo-keyfile
```

Il est important pour la suite de vérifier que l'utilisateur `mongodb` est bien le propriétaire du fichier `mongo-keyfile`, est que les droits de ce fichier sont 600.Tout ceci semble bon. Nous supprimons enfin notre manager et allons le relancer avec de nouvelles options. Notez que la commande suivante supprime le container applicatif, et que l'option -f pour force permet de forcer l'arrêt du container (autrement la suppression n'était pas possible)

```
& docker rm -f manager
```

Puis nous recréons (en relançant automatiquement) notre container en lui spécifiant quelques options supplémentaires que nous détaillerons rapidement après:

```
$ docker run --name manager --net monClusterMongo --ip 172.16.0.100 --hostname manager -
v mongo_storage_manager:/data --add-host manager:172.16.0.100 --add-host
worker1:172.16.0.101 --add-host worker2:172.16.0.102 -p 27017:27017 --entrypoint
/usr/bin/mongod -d nonoroazoro/rpi-mongo --keyFile /data/keyfile/mongo-keyfile --replSet
'rs1' --port 27017
```

- `--keyFile` précise l'endroit où se trouve le fichier contenant la clé privée. Comme dit précédemment, cette clé doit être communiquée à tous les nœuds du cluster.
- `--replSet` indique le nom du set de replication à utiliser

Normalement tout s'est bien passé et nous pouvons voir le container actif (état Up) avec la commande docker ps. Si ce n'était pas le cas, un moyen simple de diagnostiquer le problème est de lancer le container avec la commande `docker start manager –i`

L'option `–i` permet ici d'attacher les flux stdin et stdout du container à notre terminal, et donc de voir les messages d'erreur qui auraient pu être générés.

2.2.4 Configuration de mongodb sur manager

Notre nœud principal de cluster est lancé, il utilise la clé privée `mongo-keyfile`. Avant de continuer, il nous faut définir un login et un mot de passe pour le **compte administrateur** mongodb. Il s'agit du compte local. Nous utiliserons login : **roger** et mot de passe : **wilco** [1]

Nous allons désormais configurer le serveur mongodb. De manière analogue à ce que l'on ferait avec un serveur mysql, on se connecte au serveur via un client, ici `mongo`. On va simplement lancer le client mongo présent dans le nœud `manager`. Dans la suite, ce qui est en bleu clair sont les commandes à taper dans le shell mongo, et en bleu foncé le résultat du serveur affiché à l'écran.

On donne d'abord les droits d'exécution au client mongo :

```
docker exec -ti manager bash -c 'chmod +x /mongodb-rpi/mongo/bin/*'
```

```
$ docker exec -ti manager mongo
```

```
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.3
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
      http://docs.mongodb.org/
Questions? Try the support group
      http://groups.google.com/group/mongodb-user
>
```

[1] hommage à un vieux jeu vidéo qui date d'avant votre naissance

Le prompt > indique que nous sommes bien connectés au serveur mongodb, prêts à lui envoyer nos requêtes. A la première connexion, aucun compte administrateur ou sécurité n'est actif. N'importe qui peut exécuter des requêtes. Nous changerons cela par la suite. Tout d'abord, on initialise notre serveur comme faisant partie d'un cluster de réplication, en copiant les lignes suivantes dans notre client mongo :

```
> rs.initiate({
 _id: 'rs1',
 members: [{
  _id: 0, host: 'manager:27017'
 }]
})
```

Cette ligne appelle la fonction `initiate()` de la classe `rs` (pour replica set), et annonce que l'hôte `manager` fait désormais partie du cluster. Le résultat retourné à l'écran devrait être le suivant :

```
{
      "ok" : 1,
      "operationTime" : Timestamp(1520248553, 1),
      "$clusterTime" : {
            "clusterTime" : Timestamp(1520248553, 1),
            "signature" : {
                  "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                  "keyId" : NumberLong(0)
            }
      }
}
```

Puis on crée un administrateur de la base de données, en sélectionnant d'abord la base admin comme suit :

```
> admin = db.getSiblingDB("admin")
```

On ajoute alors un super utilisateur, `roger`, à qui on donne le rôle d'administrateur :

```
> rs1:PRIMARY> admin.addUser( "roger","wilco")
```

Une fois créé, nous nous authentifions en tant que `roger` sur le système :

```
admin.auth("roger", "wilco" )
```

Notez que le prompt peut à un moment changer en quelque chose comme ceci :

```
rs1:PRIMARY>
```

Ceci signifie que la réplication est désormais active, et que le nœud actuel, ici manager, est le chef de ce cluster (`PRIMARY`). Il se peut que ce statut puisse changer au fur de l'évolution de notre cluster. Généralement, les commandes de configuration du cluster ne devraient être réalisées que sur le neoud `PRIMARY`.

On verifie si le statut de note set est ok. Pour cela on va lancer la commande `rs.status()`.

```
rs1:PRIMARY> rs.status()
{
...
      "members" : [
            {
                  "_id" : 0,
                  "name" : "manager:27017",
                  "health" : 1,
                  "state" : 1,
                  "stateStr" : "PRIMARY",
            }
...
```

```
}
```

Le nœud `manager` est désormais présent dans notre set de réplication. Nous nous déconnectons et nous revenons à notre raspberry

```
exit
```

.

2.2.5 Ajout d'un noeud mongodb à notre cluster

Il ne nous reste plus qu'à configurer les nœuds worker à ajouter à notre cluster mongoDB. Pour ce faire, il nous suffit de déployer un serveur mongoDB sur chaque worker après lui avoir renseigné la clé privée, et d'indiquer au manager que le nœud est à rajouter. Après quoi, toute la configuration et les données du nœud manager vont être copiées automatiquement sur les nœuds worker.

La majorité des étapes de déploiement sur les nœuds worker ont déjà été réalisées sur le nœud manager. Il n'y a rien de nouveau. On lance donc notre nœud worker1 tout d'abord sans sécurité, avec le volume associé précédemment créé, sur le réseau monClusterMongo et avec l'IP correspondante

```
$ docker create --name worker1 -v mongo_storage_worker1:/data --net
monClusterMongo --ip 172.16.0.101 nonoroazoro/rpi-mongo

docker start worker1
```

On y crée le repertoire `/data/keyfile`

```
$ docker exec worker1 bash -c 'mkdir /data/keyfile'
```

On copie la clé privée, on change les droits et le propriétaire

```
$ docker cp mongo-keyfile worker1:/data/keyfile/
$ docker exec worker1 bash -c 'chown -R mongodb:mongodb /data'
$ docker exec worker1 bash -c 'chmod 600 /data/keyfile/mongo-keyfile'
```

Puis on supprime le containeur applicatif :

```
$ docker rm -f worker1
```

On recrée notre container `worker1` en ajoutant un nom d'hôte, les noms hôtes des autres machines du cluster, le chemin vers la clé privée, et le nom du replica set.

```
docker create --name worker1 -v mongo_storage_worker1:/data --net
monClusterMongo --ip 172.16.0.101 --hostname worker1 --add-host
manager:172.16.0.100 --add-host worker1:172.16.0.101 --add-host
worker2:172.16.0.102  --entrypoint /usr/bin/mongod nonoroazoro/rpi-
mongo  --keyFile /data/keyfile/mongo-keyfile --replSet 'rs1'
```

Enfin, on lance le containeur worker1.

```
docker start worker1
```

Nous ajoutons enfin le nœud worker1 au cluster comme suit. Nous nous nous connectons au nœud `manager` :

```
docker exec –ti manager mongo
```

Nous nous authentifions avec le compte administrateur du cluster.

```
 db.getSiblingDB("admin").auth("roger","wilco");
```

Nous ajoutons ensuite notre nœud `worker1` à l'ensemble de réplication avec la commande `rs.add()` :

```
rs1:PRIMARY> rs.add("worker1:27017")
{
```

```
        "ok" : 1,
        "operationTime" : Timestamp(1520254899, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1520254899, 1),
                "signature" : {
                        "hash" : BinData(0,"lqX8y87FDDLEKD4oLsMK9VMxERk="),
                        "keyId" : NumberLong("6529417825516257281")
                }
        }
}
```

et voilà. Notre nœud est correctement configuré et ajouté. Nous pouvons le vérifier avec la commande `rs.status()` :.

```
rs1:PRIMARY> rs.status()
...
                {
                        "_id" : 1,
                        "name" : "worker1:27017",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 23957,
…
```

Cette commande doit retourner `SECONDARY` comme statut du nœud `worker1`.

Nous effectuons à nouveau toutes ces manipulations, en remplaçant `worker1` par `worker2` (et l'adresse IP par 172.16.0.102). Nous disposons enfin d'un cluster de trois nœuds.

2.3 Ajout des données via le nœud primaire

Nous nous connections au nœud `manager` en tant que `roger`, et ajoutons une base de données `test`, une collection `collectionTest`, et quelques enregistrements.

```
docker exec –ti manager mongo
MongoDB server version: 2.4.3

rs1:PRIMARY> db.getSiblingDB("admin").auth("roger", "wilco" )
rs1:PRIMARY>  use maDBTest;

switched to db maDBTest
rs1:PRIMARY> db.createCollection("collectionTest");
{"ok" : 1}
rs1:PRIMARY>  db.collectionTest.insert({ 'bonjour:1'});
rs1:PRIMARY> db.collectionTest.insert({bonjour : 2 });
rs1:PRIMARY> db.collectionTest.insert({au revoir : 3 });
```

On vérifie que tout a bien été inséré :

```
rs1:PRIMARY> db.collectionTest.find()
{ "_id" : ObjectId("5ab23fa564a43794ea17f684"), "bonjour" : 1 }
{ "_id" : ObjectId("5ab23fac64a43794ea17f685"), "bonjour" : 1 }
{ "_id" : ObjectId("5ab23fb064a43794ea17f686"), "bonjour" : 2 }

exit
```

On vérifie enfin que les données ont été propagées sur worker1 et worker2 en se connectant dessus et en interrogeant la base. On se connecte au nœud `worker1` :

```
docker exec -ti worker1 mongo
```

```
MongoDB shell version v2.4.3
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 2.4.3
rs1:SECONDARY>
```

On s'authentifie comme `roger`

```
db.getSiblingDB("admin").auth("roger", "wilco" )
```

on autorise la consultation des données sur système secondaire :

```
rs1:SECONDARY> rs.slaveOk()
```

On liste les bases, on sélectionne notre base et effectuons une requête sur la collection donnée

```
rs1:SECONDARY> show dbs;
```

```
admin        0.000GB
config       0.000GB
maDBTest     0.000GB
local        0.001GB
```

```
rs1:SECONDARY> use maDBTest
```

```
switched to db maDBTest
```

```
rs1:SECONDARY> show collections
```

```
collectionTest
```

```
rs1:SECONDARY> db.collectionTest.find()
```

```
{ "_id" : ObjectId("5ab23fa564a43794ea17f684"), "bonjour" : 1 }
{ "_id" : ObjectId("5ab23fac64a43794ea17f685"), "bonjour" : 1 }
{ "_id" : ObjectId("5ab23fb064a43794ea17f686"), "bonjour" : 2 }
```

3 Webographie :

Cette webographie comporte les liens qui ont servi d'inspiration à la réalisation de ce TP

I. https://medium.com/towards-data-science/how-to-deploy-a-mongodb-replica-set-using-docker-6d0b9ac00e49

II. https://stackoverflow.com/questions/43733112/how-to-create-mongodb-cluster-as-docker-containers