RAPPORT DE PROJET DE FIN D'ETUDE

Software Provisioning and Automatic Deployment in a RPI Cluster

Encadré par :

Pr. LE MOUËL Frédéric

Fait par :

CHIHEB Yacine

2015-2016

Table des matières

I. Introduction

1. Contenu du rapport

Etant étudiant en Master 1 à L'INSA Lyon, j'ai eu l'opportunité de réaliser mon projet de fin d'études au sein du Laboratoire CITI.

L'objectif de mon projet de fin d'études est de mettre en place un cluster de Raspberry Pi, analyser les solutions disponibles et effectuer des tests de performances.

Pour cela, j'ai fait l'état des lieux des technologies utilisées pour permettre la communication entre les objets connectés plus spécifiquement la technologie Docker.

Dans un premier temps, je vais présenter comment les objets connectés ont révolutionnés le monde.
Ensuite, je vais présenter le contexte dans lequel le projet a été initié.
Par la suite, je vais aborder la problématique que je vais traiter durant mon rapport.
Suite à cela, je vais décrire le déroulement du projet, proposer des alternatives pour optimiser le processus de mise en conformité.
Enfin, j'apporterai une conclusion qui résumera les idées discutées dans ce rapport.

2. Contexte du projet

Internet est irréprochablement l'une des inventions les plus significatives de toute l'histoire de l'humanité, mais l'émergence de ce qu'on appelle Internet of Things a changé la donne et a fait révolutionner le monde. Il s'agit maintenant de connexions et intéractions multiples entre machines, personnes et processus.

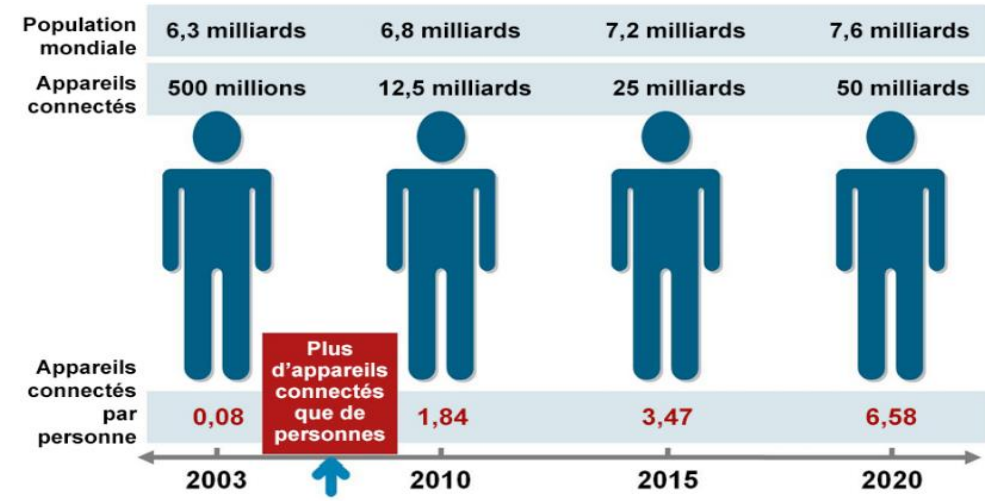
Un objet, un composant intelligent, de la connectivité et des données en masse, tous ces éléments vont être conjugués pour envahir notre quotidien avec une vitesse exceptionnelle.
Cette révolution touche tous les secteurs d'activités; pour le secteur industriel, par exemple, la possibilité de relier des capteurs en réseau et d'établir des communications machine to machine(M2M) redéfinit la conception de la chaine de production. De plus l'IoT peut servir de lien entre les PGI(ERP), la gestion de la chaîne d'approvisionnement et les outils de gestion de la relation client.

C'est important de définir ce concept, selon l'Union Internationale des Télécommunications, "Internet of Things" est une :

« infrastructure mondiale pour la société de l'information, qui permet de disposer de services évolués en interconnectant des objets (physiques ou virtuels) grâce aux technologies de l'information et de la communication interopérables existantes ou en évolution »

Ce concept est né au Massachusets Institue of Technologie (MIT), et plus particulièrement au groupe Auto-ID Center. Ce groupe, formé en 1999, travaillait sur l'identification de la fréquence radio (RFID) a conçu l'architecture de l'IoT en collaboration avec ECPGlobal.

Depuis, le nombre d'appareils connectés n'a cessé d'évoluer, avec des estimations allant jusqu'à 50 milliards en 2020.



Source : Cisco IBSG, avril 2011

Cependant, un problème surgit face à cette croissance exponentielle d'objets connectés, en effet le défi majeur de l'IoT est la gestion du volume de données engendrées. Pour illustrer ce phénomène, les communications humaines depuis la nuit des temps jusqu'à 2003 est l'équivalent en données numériques à 48h aujourd'hui. Il s'agit donc de mettre en place de nouvelles techniques de traitement de données.

Les objets connectés produisent donc un flux constant de données qui peuvent être utilisés pour améliorer nos activités personnelles ou professionnelles. Il est intéressant de constater que tout le problème est là : un flux de données 24/7, étant donné le faible coût des capteurs le volume d'information explose ; un capteur qui génère une mesure toutes les secondes, cela fait 31,5 million de valeurs par an. A titre d'exemple, un Airbus A380 contient environs 300 000 capteurs ce qui fait 16 To par vol.

Les bases de données classiques ne sont pas conçues pour stocker, consulter et analyser ce tel volume de données, il s'agit maintenant de mettre en place des solutions qui permettent d'offrir des services pour le stockage et traitement des données de façon à optimiser les ressources disponibles .Un nouveau concept surgit: le cloud computing.

Le cloud computing ouvre une nouvelle perspective à cet environnement en proposant des solutions en ligne, à la demande, disponible à tout moment et pour n'importe quel terminal.
En bref, le cloud computing permet de partager, via un fournisseur d'offres Cloud, une infrastructure, une solution applicative ou encore une plateforme à tout utilisateur via internet.
Aujourd'hui, le Cloud non seulement propose la possibilité de disposer d'un espace de stockage en ligne, mais aussi une puissance de calcul externe aux dispositifs.

Les avantages qu'offre l'utilisation de la solution Cloud Computing sont nombreux :

- Un usage simplifié; l'utilisateur n'a pas d'infrastructure à gérer, c'est au fournisseur Cloud de maintenir le matériel serveur, le stockage et le réseau.
- Une réduction des coûts; il permet de démarrer une activité professionnelle sans avoir à investir dans une infrastructure IT très coûteuse.
- Haute disponibilité des services ; les utilisateurs ont accès à leurs applications et donnés sans interruption des services.
- La sécurité ; les fournisseurs Cloud garantissent une sécurité en possédant des dispositifs et systèmes de sécurité qui sont régulièrement mis à jour.

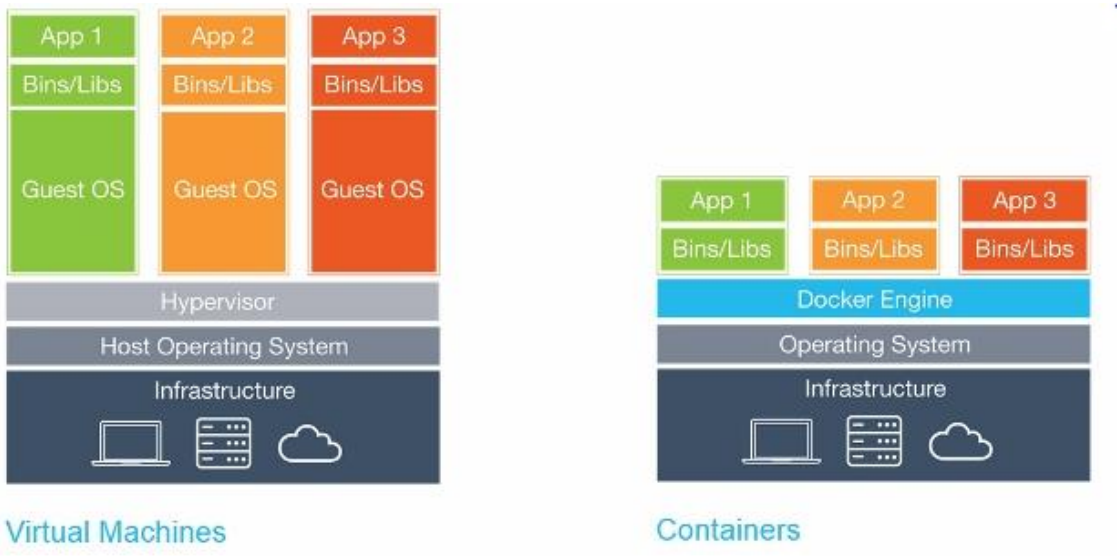
Les fournisseurs de Cloud permettent généralement de créer des machines virtuelles à partir d'images prédéfinies, mais ces images de systèmes d'exploitation génériques ne répondent pas toujours aux besoins d'une entreprise. Par conséquent, ils doivent offrir la possibilité de créer et d'exploiter des images personnalisées de machines virtuelles.

La virtualisation traditionnelle permet, via un hyperviseur, de simuler une ou plusieurs machines physiques, et les exécuter sous forme de machines virtuelles (VM) sur un serveur machine. Ces VM intègrent elles-mêmes un OS sur lequel les applications qu'elles contiennent sont exécutées.

Cependant, cette solution met en évidence un nouveau problème, la compatibilité entre les différents OS, en effet, une application conçue sur Linux ne peut être exécuté sur Windows. Cela est particulièrement contraignant à partir du moment où une entreprise veut changer de fournisseur Cloud.

Une nouvelle technologie voit donc le jour en 2013, appelé Docker. Cette dernière permet de basculer une application entre les clouds, notamment Amazon, Google et Microsoft.
Docker se base sur une virtualisation par container. En effet, une application embarquée dans un container virtuel peut être exécuté sur n'importe quel serveur car il fait appel directement à l'OS de la machine hôte sans mettre en place un nouvel OS. Étant donné que Docker est basé sur Linux, il peut donc être facilement déplacé d'une machine à l'autre.

Virtual Machines VS Containers



En plus de la portabilité que permet l'utilisation des containers, grâce à ce système, le déploiement des images est beaucoup plus léger. La mise en place d'une machine virtuelle en utilisant un hyperviseur peut peser plusieurs Go, alors qu'avec Docker, ce ne sont que quelque Mo. Cela permet d'économiser en espace et optimiser les ressources en augmentant notamment la vitesse de traitement.

Ces avantages permettent en plus une facilité de déploiement qui est très appréciée pour effectuer des tests de nouvelles architectures. C'est donc Docker qui va être utilisé pour la mise en place d'un cluster de Raspberry Pi, en testant différents contextes de performances et de fiabilités du système.

II. Choix de la solution à Adopter

L'objectif du projet est d'effectuer des tests de performances d'un cluster de Raspberry Pi. Les tâches qui devront être implémentées sont :

- La mise en place un cluster, l'outil qui va être utilisé pour cela est Docker Swarm.
- La jointure des RPI au cluster grâce à Docker Machine.
- Le déploiement d'une application multiservice en se servant de Docker Compose.

a) Raspberry Pi

Le dispositif qui sera utilisé pour mettre en place le cluster est la Raspberry Pi 2 Model B, il est donc nécessaire de détailler les spécifications de cet outil.

Pour les besoins du projet, le choix de l'utilisation des Raspberry Pi est judicieux non seulement en terme de consommation d'énergie car c'est un outil à très faible consommation, mais aussi parce qu'elle permet la mise en place d'un serveur, un NAS, ou encore pour en faire un media-center.
En termes de performances, la RPI 2 propose 1Go de RAM, et est composée d'un processeur de 4 cœurs, ce qui pourrait même permettre la mise en place d'un Windows 10.

b) Docker et Raspberry Pi

L'utilisation de Docker pour le déploiement des containers bouleverse la méthode traditionnelle de la virtualisation des machines. Cependant un problème majeur se pose quand à la compatibilité avec les Rasperry, en effet, chaque Dockerfile comprend une image de base. L'exécution des applications ont besoin d'une machine de base x86, alors que la RPI est conçu sur la technologie ARM. Cela est particulièrement contraignant car il n'est pas possible d'exécuter des applications conçues en x86 sur les RPI.

En raison de fort déploiement de terminaux conçus en technologie ARM, notamment grâce à la faible consommation d'énergie, des communautés se sont mobilisé pour mettre en place une adaptation de Docker à l'architecture ARM, dont la plus importante est Hypriot.
En Février 2014 Hypriot, développe une première version d'OS basé sur une version minimal de Debian.

Les caractéristiques les plus importantes de HypriotOS sont :

- Un OS basé sur Debian.
- Parfaitement optimisé pour l'utilisation de Docker
- Mise à jour en amont avec les versions officielles de Docker.
- Facilité de téléchargement, flash et boot.

c) Docker Swarm

Docker dispose d'un système appelé Swarm qui permet de gérer un ensemble d'hôtes comme un pool de ressources unique. Cela est particulièrement intéressant lorsque le nombre de Raspberry augmente considérablement. Cela permet donc de disposer d'un système distribué qui permet de répartir un calcul ou un traitement sur plusieurs nœuds, ce système constitué de plusieurs nœuds sera considéré comme une unité centrale.

Swarm utilise des techniques de planification (scheduling) pour s'assurer d'une disponibilité des ressources suffisantes pour les containers distribués. Il attribue des containers aux nœuds et optimise les ressources en programmant les charges de travail sur l'hôte le plus approprié. Cette orchestration permet donc d'équilibrer les containers en s'assurant qu'ils sont lancés sur des systèmes disposant de ressources suffisantes, tout en maintenant les niveaux de performance requis.

Swarm dispose de trois stratégies différentes pour déterminer sur quels nœuds chaque container doit être exécuté :

- **Spread** - C'est la stratégie réglée par défaut. Elle permet un équilibre des containers sur les nœuds d'un cluster en se basant sur la disponibilité CPU et de RAM, ainsi que sur le nombre de containers sur chaque nœud. L'avantage principal de cette stratégie est que dans le cas où un nœud tombe, le nombre de containers perdus est relativement réduit.

- **Binpack** - L'intérêt de cette stratégie est d'utiliser pleinement chaque nœud. Une fois un nœud est plein, il passe au suivant dans le cluster. L'avantage de Binpack est qu'il garde des ressources en réserve et laisse plus d'espace pour les grands containers sur les machines inutilisées.

- **Random** - Choisit un nœud au hasard.

d) Docker Machine

Pour la gestion des nœuds du cluster, Docker dispose d'une commande appelé Docker Machine. Grâce à cette dernière il est possible de joindre, arrêter, redémarrer et connaître l'état d'un nœud à travers la CLI docker-machine.
Il est possible d'établir une hiérarchie des nœuds en spécifiant lesquelles aurons le contrôle sur les autres, il y a donc un nœud master et des nœuds slaves.

e) Docker Compose

L'exécution d'images avec Docker est particulièrement simple, avec la commande run il est possible de lancer une image. Cependant il arrive souvent qu'une application nécessite l'exécution de plusieurs images pour fonctionner. Ce qui rend la tâche de lancer chaque image séparément assez contraignante.
Docker dispose d'un outil pour faciliter cette tâche : Docker Compose.

Compose permet donc la définition et l'exécution d'une application multi-containers avec une seule commande.
Pour cela, il est possible d'utiliser un fichier Compose pour configurer les services dont l'application a besoin. Puis, en utilisant une seule commande, de créer et démarrer tous les services en même temps.

L'utilisation de Compose est un processus en trois étapes :

- Définir un environnement de l'application avec un Dockerfile.
- Définir les services qui composent l'application dans un fichier texte "docker-compose.yml". Cela permet aux services d'être exécutés ensemble dans un environnement isolé.
- Lancement de l'application en utilisant la commande docker-compose up.

III. Installation de l'OS

La première étape est l'installation et configuration de l'OS sur la carte micro SD.
La version de l'OS qui sera utilisés est **HypriotOS 0.7.0 Berry (beta)**. Les caractéristiques sont :

- Linux kernel 4.1.17
- Docker Engine 1.10.2
- Docker Compose 1.6.2
- Swarm 1.1.3

Pour l'installation de l'OS différentes manières sont possibles selons les choix de l'utilisateur. Sur Windows une application est nécessaire pour flasher la carte SD, par exemple "rufus". Sur Linux, il est possible de flasher la carte SD directement sur le terminal.

a) Le réseau

Les RPI seront connectés en mode étoile. C'est en se basant sur les adresse IP que la configuration du cluster va être faite. Il est préférable de disposer d'une adresse IP statique pour éviter la reconfiguration du système.

Plusieurs manières d'accéder au réseau sont possibles :

- À travers le serveur DHCP du routeur ou la "box" internet. Cette configuration a comme inconvénient une allocation dynamique de l'adresse IP. Aillant a reconfigurer le cluster en cas de changement d'adresse.

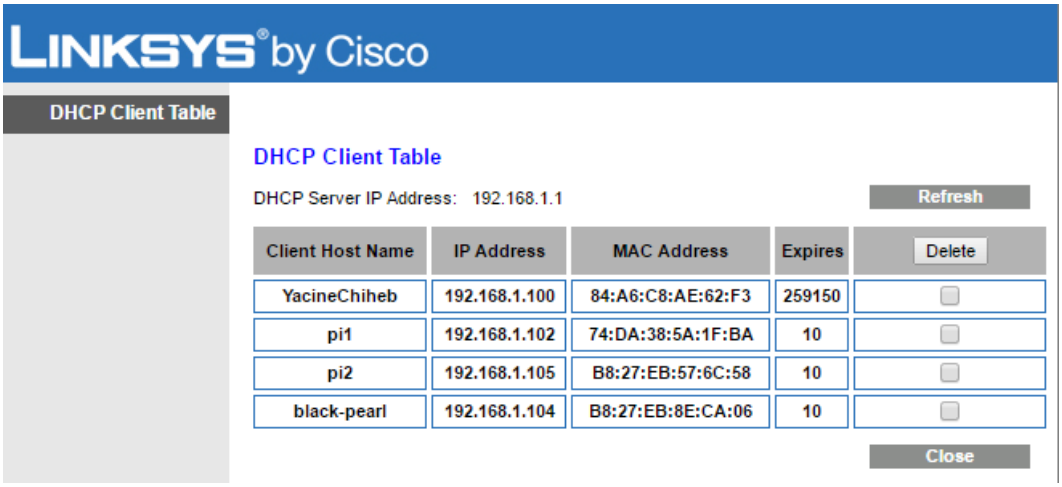
- En allouant une adresse statique à la RPI, pour cela, une configuration est nécessaire.

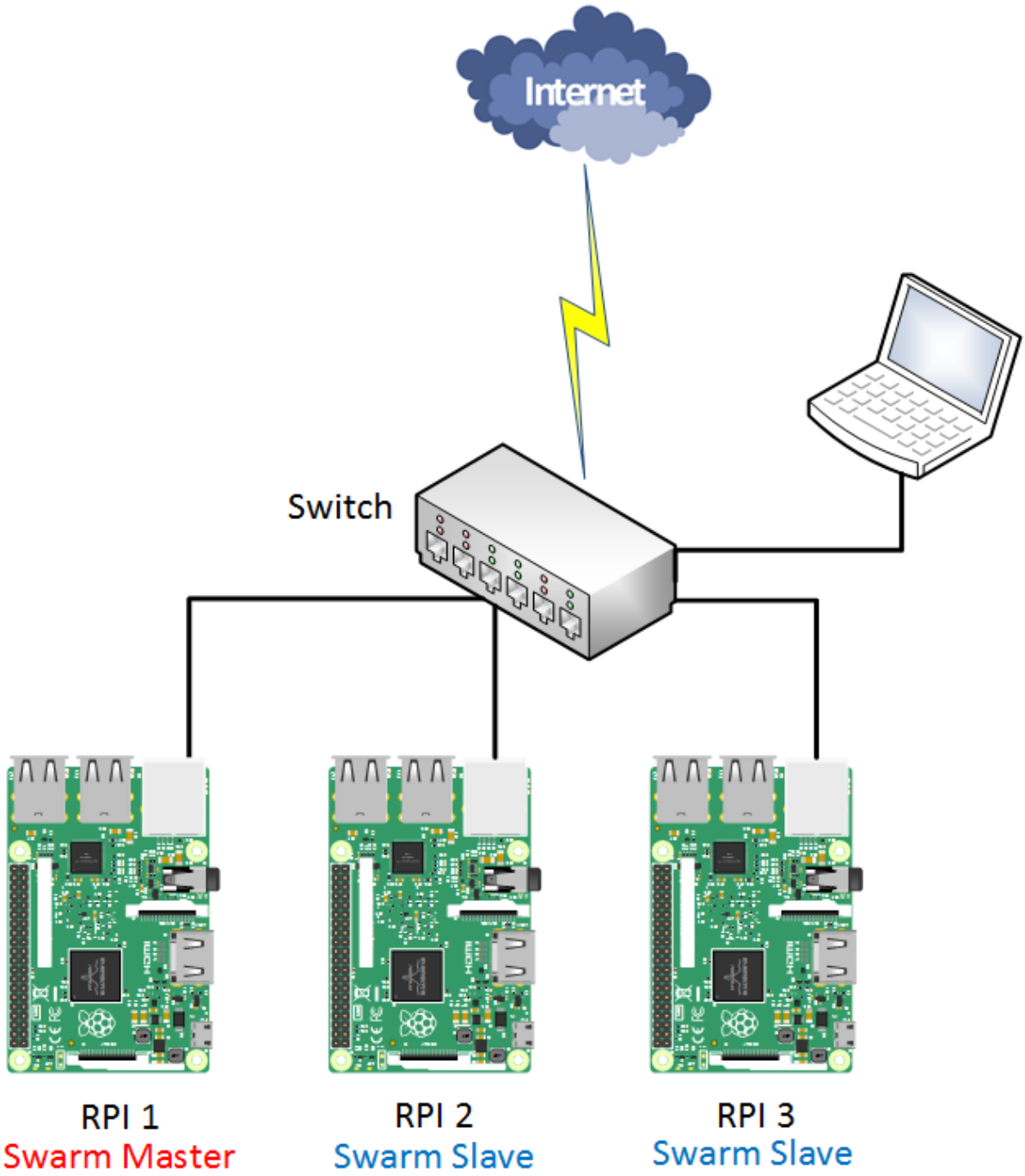
    - Avec un éditeur texte comme "vi" on accède au fichier /etc/network/interfaces.
    - Remplacer la ligne "iface eth0 inet dhcp" par "iface eth0 inet static".
    - Introduire les paramètres du réseau ; address, netmask, network, gateway.
    - Enregistrer et redémarrer la RPI.

    Une configuration du serveur DNS est aussi nécessaire.

    - Accéder au fichier /etc/resolv.conf
    - Ajouter les lignes :
        - nameserver 8.8.8.8 #adresse publique du serveur DNS de Google
        - nameserver 8.8.4.4

- Pour se connecter en mode Wifi, il faut utiliser une clef usb Wifi. La version de l'OS utilisé dispose d'un fichier de configuration "occidentalis.txt". Dans ceci il faut introduire les paramètres du réseau wifi; nom et mot de passe.


LINKSYS by Cisco
DHCP Client Table
DHCP Client Table
DHCP Server IP Address: 192.168.1.1
Refresh
Client Host Name
IP Address
MAC Address
Expires
Delete
YacineChiheb
192.168.1.100
84:A6:C8:AE:62:F3
259150
pi1
192.168.1.102
74:DA:38:5A:1F:BA
10
pi2
192.168.1.105
B8:27:EB:57:6C:58
10
black-pearl
192.168.1.104
B8:27:EB:8E:CA:06
10
Close


Internet
Switch
RPI 1
Swarm Master
RPI 2
Swarm Slave
RPI 3
Swarm Slave

b) Generation des keygen

L'accès au RPI se fera grâce au protocole ssh (Secure Shell). Une fois la RPI connecté au réseau il est possible donc d'y accéder avec la commande "ssh *adresse_IP_RPI*". L'authentification des RPI slaves par la RPI master peut être faite avec la génération d'une clé publique qui est distribué sur les composants du cluster pour permettre un accès sans utiliser de mot de passe :

ssh-keygen -t rsa -R $IP_ADDRESS

c) Configuration du Cluster

Une fois les RPI connectés au réseau il est désormais possible de configurer le cluster. Les étapes pour la mise en place du cluster sont :

- Génération du TOKEN ; pour que tous les agents du cluster puissent communiquer entre eux, Docker Swarm utilise un identifiant (TOKEN) qui est partagé par les RPI. La commande est :

export TOKEN=$(for i in $(seq 1 32); do echo -n $(echo "obase=16; $(($RANDOM % 16))" | bc); done; echo)

NB: La commande "bc" n'étant pas disponible par défaut sur l'OS utilisé, un simple "apt-get install bc" permet de l'installer.

- Création du swarm master sur une des RPI; la commande Docker est:

```
docker-machine create -d hypriot \
    --swarm --swarm-master \
    --swarm-discovery token://$TOKEN \
    --hypriot-ip-address $IP_ADDRESS $1
```

- Création de tous les agents à joindre au cluster en mode slave:

```
docker-machine create -d hypriot \
    --swarm \
    --swarm-discovery token://$TOKEN \
    --hypriot-ip-address $IP_ADDRESS $1
```

- Pour se connecter au cluster, la commande est la suivante:

eval $(docker-machine env --swarm "hostname_RPI")

- Pour visualiser le cluster:

docker info

d) Scripts

Pour un déploiement plus efficace du cluster, j'ai implémenter les scripts suivants:

- **getip**

```
#!/bin/bash

IP_AD=$( traceroute $1 2>&1 | head -n 1 | cut -d\( -f 2 | cut -d\) -f 1 )

if [[ $IP_AD =~ 127.* ]]
then
      IP_AD=$( ifconfig wlan0 | head -n 2 | cut -d" " -f12-13 | cut -d\: -
f2 )
fi

echo $IP_AD
```

- **configuration _ssh**

```
#!/bin/bash

IP_ADDRESS=$(./getip $1)

echo $IP_ADDRESS

if [ ! -e ~/.ssh/id_rsa ]
then
      if [ ! -e ~/.ssh/known_hosts ]
      then
            ssh-keygen -t rsa -R $IP_ADDRESS
      else
            ssh-keygen -t rsa
      fi
fi

ssh-copy-id -i ~/.ssh/id_rsa.pub root@$1
```

- **configuration_master**

```
#!/bin/bash

IP_ADDRESS=$( ./getip $1 )

echo $IP_ADDRESS

TOKEN=66DE21F043CC208A267FCB4CD8BACAA3
echo $TOKEN

docker-machine create -d hypriot --swarm --swarm-master --swarm-discovery
token://$TOKEN --hypriot-ip-address $IP_ADDRESS $1
```

- **configuration_slave**

```
#!/bin/bash

IP_ADDRESS=$( ./getip $1 )

echo $IP_ADDRESS

TOKEN=66DE21F043CC208A267FCB4CD8BACAA3
echo $TOKEN

docker-machine create -d hypriot --swarm --swarm-discovery token://$TOKEN -
-hypriot-ip-address $IP_ADDRESS $1
```

e) Docker Compose

Le choix de l'application multi-services à tester est *Wordpress.*
Pour cela, il faut cloner le répertoire rpi-wordpress sur la RPI master :

git clone https://github.com/rothgar/rpi-wordpress

Les containers utilisés sont :

- rothgar/rpi-caddy :0.9.1 - Serveur Web
- rothgar/rpi-php :5.6 - PHP Engine
- rothgar/rpi-mysql :5.5 - Base de données

Une fois les containers disponibles sur la RPI, il est temps de télécharger l'application Wordpress :

curl –sL http://wordpress.org/latest.tar.gz | tar --strip 1 –xz –C .data/wp/www

Apres un paramétrage du container de la base de données mysql :

```
version: '2'
services:
  caddy:
    image: rothgar/rpi-caddy:0.9.1
    #build: caddy
    volumes:
      - "./.data/wp:/srv"
    depends_on:
      - mysql
      - php
    ports:
      - "80:80"
      - "443:443"
    restart: always
```

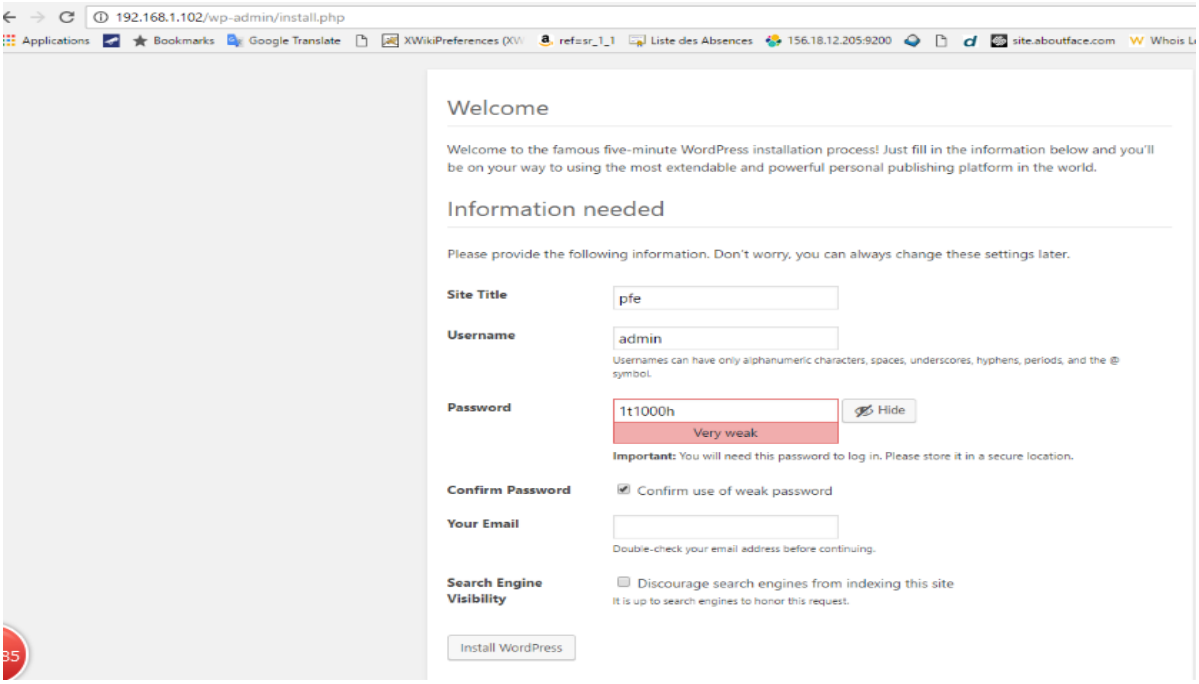
```
php:
  image: rothgar/rpi-php:5.6
  #build: php
  links:
    # make sure you change your wp-config
    - mysql:mysql
  volumes:
    - "./.data/wp:/srv"
  restart: always
```

```
mysql:
  image: rothgar/rpi-mysql:5.5
  #build: mysql
  volumes:
    - "./.data/db:/var/lib/mysql"
    - "./.data/backup/1n7:/docker-entrypoint-initdb.d"
  restart: always
  environment:
    # Change for your existing database
    MYSQL_ROOT_PASSWORD: wordpress
    # below values should match wp-config
    MYSQL_DATABASE: blog
    MYSQL_USER: wordpress
    MYSQL_PASSWORD: 1t1000h
```

Lancement du service *docker-compose up –d* :

```
HypriotOS: root@pi2 in ~/rpi-wordpress on master*
$ docker-compose up -d
Starting rpiwordpress_mysql_1
Starting rpiwordpress_php_1
Starting rpiwordpress_caddy_1
HypriotOS: root@pi2 in ~/rpi-wordpress on master*
```

Pour la confirmation de la bonne installation :



Cependant un problème survient au moment de la configuration du cluster avec swarm. La version utilisé ne permet pas d'établir le lien entre docker swarm et docker compose du au faite de la nécessité de créer un réseau privé qui englobe les containers du service wordpress.

```
HypriotOS: root@pi2 in ~
$ docker network create -d overlay mynet
Error response from daemon: failed to parse pool request for addre
ss space "GlobalDefault" pool "" subpool "": cannot find address s
pace GlobalDefault (most likely the backing datastore is not confi
gured)
HypriotOS: root@pi2 in ~
```

IV. Solution alternative

a) Consul

Pour solutionner ce problème, une configuration alternative est nécessaire : Utilisation de docker compose pour le lancement de docker swarm. En mettant en place une interface graphique : consul, pour visionner le cluster.

```
HypriotOS: root@pi2 in ~
$ cat docker-compose.yml
  consul:
    image: "hypriot/rpi-consul"
    ports:
     - "8500:8500"
    command: "agent -server -data-dir /data -bootstrap-expect 1 -ui-dir /ui -cl
ient=0.0.0.0"
    volumes:
     - "/data"

HypriotOS: root@pi2 in ~
```

Avant tout il faut exposer chaque nœud à un réseau commun. Par défaut, Engine écoute sur un fichier de socket local appelé /var/run/docker.sock, mais pour que les nœuds soient a l'écoute sur le même réseau, il faut les exposer sur un port commun qui est configuré par le nœud master. Pour cela il faut ajouter sur le fichier appelé '/etc /default /docker' sur chaque nœud :

DOCKER_OPTS='-H unix:///var/run/docker.sock -H tcp://0.0.0.0 :2375

Cette ligne permet de continuer à utiliser le socket local, mais s'expose également sur le port 2375 sur toutes les interfaces réseau. Il faut alors redémarrer le service docker : *service docker restart* pour appliquer cette configuration et répété cela sur chaque nœud.

b) Swarm

Pour joindre les nœuds au cluster il faut lancer le container client swarm sur chaque nœud client, puis un container master sur le nœud master.

```
HypriotOS: root@pi2 in /var/master
$ cat docker-compose.yml
swarm:
  image: "hypriot/rpi-swarm"
  ports:
   - "1234:2375"
  command: "manage consul://192.168.1.102:8500/swarm"
```

```
HypriotOS: root@pi2 in /var/client
$ cat docker-compose.yml
swarm:
  image: "hypriot/rpi-swarm"
  command: "join --addr=192.168.1.102:2375 consul://192.168.1.102:8500/swarm"
  restart: always
HypriotOS: root@pi2 in /var/client
```

Cela a créé une interface swarm qui a ensuite été exposé sur le port 1234 du nœud master. Pour vérifier que tout fonctionne, et donc visualiser la lister des nœuds connectés au cluster :

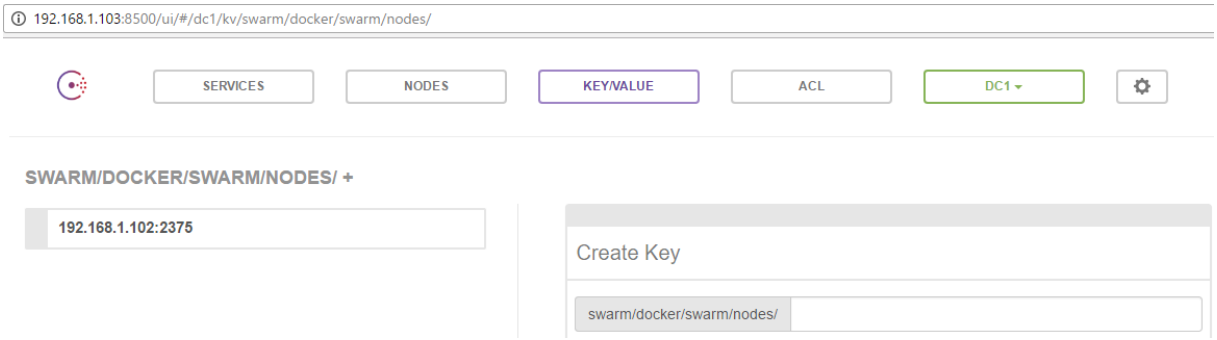
>> docker -H tcp://localhost :1234 info

```
 black-pearl: 192.168.1.104:2375
  └ ID: 4IYQ:YLSV:23EK:6G7G:DFNG:DAJI:ZU7B:XBDA:26R
  └ Status: Healthy
  └ Containers: 1
  └ Reserved CPUs: 0 / 1
  └ Reserved Memory: 0 B / 455.9 MiB
  └ Labels: executiondriver=native-0.2, kernelversi
essie), storagedriver=overlay
  └ UpdatedAt: 2016-09-21T21:16:28Z
  └ ServerVersion: 1.10.2
pi2: 192.168.1.102:2375
  └ ID: KSRI:CANK:UTYN:GJTL:5TCQ:FMWQ:RTDF:RB2S:RVJ
  └ Status: Healthy
  └ Containers: 9
  └ Reserved CPUs: 0 / 4
  └ Reserved Memory: 0 B / 971.8 MiB
  └ Labels: executiondriver=native-0.2, kernelversi
 (jessie), storagedriver=overlay
  └ UpdatedAt: 2016-09-21T21:16:29Z
  └ ServerVersion: 1.10.2
Plugins:
 Volume:
 Network:
Kernel Version: 4.1.17-hypriotos-v7+
Operating System: linux
Architecture: arm
CPUs: 5
Total Memory: 1.394 GiB
Name: 2bfa762240ab
```

c) Verification des resultats

Lancement sur le navigateur de l'adresse IP du master sur le port 8500:

192.168.1.103:8500



Sur cette image on peut apercevoir le nœud client crée dans le cluster du swarm.

d) Overlay Network

Le réseau overlay est un réseau superposé un peu comme un VPN. L'intérêt de ce réseau est de couvrir l'ensemble du cluster. Chaque conteneur est doté d'une interface supplémentaire, mais ils sont tous dans le même sous-réseau. Le trafic entre ces interfaces est canalisé entre les nœuds, et par conséquent, chaque conteneur semble être sur le même réseau de conteneurs tout en étant sur des hôtes différents. Pour cela, les nœuds doivent être capables d'accéder l'un à l'autre.

Il faut donc ajouter des d'options pour les DOCKER_OPTS dans /etc /default/docker sur chaque nœud pour leur dire d'utiliser l'instance de consul pour le clustering de réseau.

```
DOCKER_OPTS=' -H unix:///var/run/docker.sock -H tcp://0.0.0.0:2375 --cluster-store
=consul://<master IP address>:8500 --cluster-advertise=eth0:2375 '
```

Encore une fois, redémarrer le service docker sur chaque nouveau nœud : service docker restart.

Pour créer le réseau overlay, il faut exécuter la commande suivante sur le nœud master :

```
docker -H tcp://localhost:1234 network create --driver overlay my-net
```

```
HypriotOS: root@black-pearl in ~/client
$ docker network ls
NETWORK ID          NAME                DRIVER
fab47b280e7c        my_net              overlay
26dfdb30b2fe        bridge              bridge
02cb96d96664        none                null
0a18abaf46a2        host                host
```

e) Mise en réseau des containers

Une fois le réseau overlay disponible, il est temps de déployer les containers sur le sous-réseau.

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
rothgar/rpi-caddy   0.9.1               dca58140a5c2        2 weeks ago         136.1 MB
rothgar/rpi-php     5.6                 162098e62738        2 weeks ago         172.5 MB
rothgar/rpi-mysql   5.5                 438099240eb0        3 weeks ago         215.2 MB
hypriot/rpi-consul  latest              879ac05d5353        4 months ago        19.71 MB
hypriot/rpi-swarm   latest              c298de062190        6 months ago        13.27 MB
$ $ docker -H tcp://localhost:1234 run -itd --net=my_net rothgar/rpi-caddy
-bash: $: command not found
HypriotOS: root@pi1 in ~/rpi-wordpress on master*
$  docker -H tcp://localhost:1234 run -itd --net=my_net rothgar/rpi-caddy
docker: Error response from daemon: Tag latest not found in repository docker.io/rothgar/rpi-cad
dy.
See 'docker run --help'.
HypriotOS: root@pi1 in ~/rpi-wordpress on master*
$ docker -H tcp://localhost:1234 run -itd --net=my_net rothgar/rpi-caddy:0.9.1
410055733a36c57067b407a1b610fa1562cb8e1497b1ba416fbc1b5562f6de12
HypriotOS: root@pi1 in ~/rpi-wordpress on master*
$ docker -H tcp://localhost:1234 run -itd --net=my_net rothgar/rpi-php:5.6
d654f8e378ec542f93ac1e3124894169635fc0bddf29c2166e627b9f44fb8004
HypriotOS: root@pi1 in ~/rpi-wordpress on master*
$ docker -H tcp://localhost:1234 run -itd --net=my_net rothgar/rpi-mysql:5.5
b1d4715895fe11677666828333353361954eafb3b6b141236e7d2b72d265c057
```

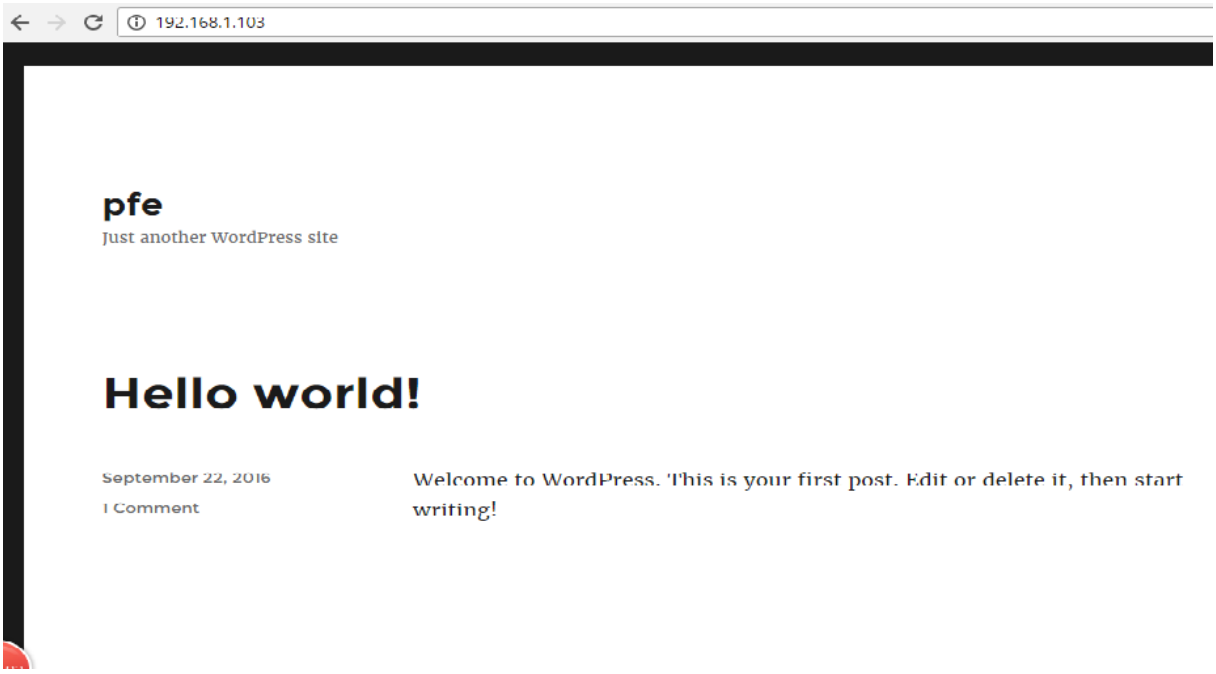
Le résultat obtenu en mettant sur le sous-réseau les containers est le déploiement par le nœud master sur le nœud client disponible. En effet le master a comme mission le déploiement des containers sur le différent nœud en se basant sur la stratégie spread pour le placement des containers.

```
$ docker -H tcp://localhost:1234 info
Containers: 5
 Running: 3
 Paused: 0
 Stopped: 2
Images: 4
Server Version: swarm/1.1.3
Role: primary
Strategy: spread
Filters: health, port, dependency, affinity, constraint
Nodes: 1
 black-pearl: 192.168.1.102:2375
  └ Status: Healthy
  └ Containers: 5
  └ Reserved CPUs: 0 / 4
  └ Reserved Memory: 0 B / 971.8 MiB
  └ Labels: executiondriver=native-0.2, kernelversion=4.1.17-hypriotos-v7+, operatingsystem
=Raspbian GNU/Linux 8 (jessie), storagedriver=overlay
  └ Error: (none)
  └ UpdatedAt: 2016-09-22T21:26:26Z
Plugins:
 Volume:
 Network:
Kernel Version: 4.1.17-hypriotos-v7+
Operating System: linux
Architecture: arm
CPUs: 4
Total Memory: 971.8 MiB
Name: 1a934ecc25bd
HypriotOS: root@pi1 in ~/rpi-wordpress on master*
```

Ce résultat obtenu démontre que le master agi sur le cluster uniquement en mode gestion, et non pas en mode client.

```
HypriotOS: root@black-pearl in ~/client
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
rothgar/rpi-caddy   0.9.1               dca58140a5c2        2 weeks ago
136.1 MB
rothgar/rpi-php     5.6                 162098e62738        2 weeks ago
172.5 MB
rothgar/rpi-mysql   5.5                 438099240eb0        3 weeks ago
215.2 MB
hypriot/rpi-swarm   latest              c298de062190        6 months ago
13.27 MB
```

Cette dernière image montre l'interface du site web crée sur l'adresse IP du nœud master, tout en étant les containers sur le nœud client.



V. Conclusion

J'ai été amené à faire l'état des lieux des matériels et des technologies utilisés pour la réalisation de mon projet.

Certes cela n'a pas été facile au debout car je n'avais pas un profile technique qui allait me permettre de comprendre assez rapidement le fonctionnement des Raspberry Pi et de Docker. Mais avec le temps, j'ai commencé à avoir les bonnes pratiques pour savoir debugger et réussir à comprendre les problèmes liés au système.

Cela m'a permis de comprendre comment créer un cluster avec des Rapsberry Pi en utilisant Docker.

Étant Docker une technologie non conçue pour les systèmes basés sur ARM, j'ai était confronté à de nombreux problèmes de compatibilités. Notamment pour le paramétrage de docker swarm.

VI. Annexes

https://docs.docker.com/

http://blog.hypriot.com/

https://github.com/

http://raspberryzero.com/

https://www.raspberrypi.org/

www.rightscale.com

www.silicon.fr

http://www.informaticslab.co.uk/infrastructure/2015/12/09/raspberry-pi-docker-cluster.html

https://medium.com/@rothgar/wordpress-in-docker-on-a-raspberry-pi-149b4301195#.bgela1d5j