Version : **2022.01**

Dernière mise-à-jour : 2021/12/29 10:32

DOF202 - Docker Compose, Docker Machine et Docker Swarm

Contenu du Module

LAB #1 - Docker Compose

Docker Compose est un utilitaire de compilation d'images et de gestion de conteneurs multiples, tous intégrés dans une seule et unique application logicielle. Son rôle est de rendre plus aisée la manipulation d'éléments multiples interconnectés. Pour ce faire, Docker Compose utilise un fichier dénommé **docker-compose.yml** au format **YAML**.

Ce fichier, appelé par la commande **docker-compose build** commence avec un mot clef **image** ou **build** selon que l'image est récupérée sur un registre ou provient du répertoire cité dans le fichier. Le reste du fichier contient des instructions pour définir la compilation des images constituantes, pour lier des conteneurs et pour définir l'environnement.

Une fois totalement construite, l'application peut ensuite être pilotée très simplement par l'utilisation de la commande docker-compose qui réagit de la même manière que la commande **docker** mais cette fois-ci sur tous les conteneurs définis dans le fichier **docker-compose.yml**.

De cette façon il est possible de démarrer l'application avec la commande **docker-compose up**, de l'arrêter avec la commande **docker-compose stop** ou de la redémarrer avec la commande **docker-compose restart**. De la même manière que la commande **docker**, la commande docker-compose donne accès aux journaux grâce à la commande **docker-compose logs**.

1.1 - Installation

Récupérez docker-compose avec **curl** :

```
root@debian9:~# curl -L "https://github.com/docker/compose/releases/download/1.23.2/docker-compose-$(uname -s)-
$(uname -m)" -o /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   617    0   617    0     0    912      0 --:--:-- --:--:-- --:--:--   912
100 5140k  100 5140k    0     0  2145k      0  0:00:02  0:00:02 --:--:-- 5232k

root@debian9:~# chmod +x /usr/local/bin/docker-compose
```

Rendez **/usr/local/bin/docker-compose** exécutable :

```
root@debian9:~# ls -l /usr/local/bin/docker-compose
-rw-r--r-- 1 root staff 5263681 Jan  2 16:29 /usr/local/bin/docker-compose
root@debian9:~# chmod u+x /usr/local/bin/docker-compose
```

Avant de commencer, installez l'utilitaire **tree** :

```
root@debian9:~# apt-get install tree
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  tree
0 upgraded, 1 newly installed, 0 to remove and 99 not upgraded.
Need to get 45.9 kB of archives.
After this operation, 102 kB of additional disk space will be used.
Get:1 http://ftp.fr.debian.org/debian/ jessie/main tree amd64 1.7.0-3 [45.9 kB]
Fetched 45.9 kB in 0s (429 kB/s)
Selecting previously unselected package tree.
(Reading database ... 100654 files and directories currently installed.)
Preparing to unpack .../tree_1.7.0-3_amd64.deb ...
Unpacking tree (1.7.0-3) ...
Processing triggers for man-db (2.7.0.2-5) ...
Setting up tree (1.7.0-3) ...
```

1.2 - Utiliser docker-compose

Une Application Simple

Dans ce LAB vous allez créer une application simple ayant deux environnements différents :

- development
- production

afin d'utiliser deux configurations différentes selon l'environnement désiré.

Commencez par créer l'arborescence du projet :

```
root@debian9:~# mkdir -p MyApp/common
root@debian9:~# mkdir -p MyApp/development/content
root@debian9:~# mkdir -p MyApp/production/content
root@debian9:~# touch MyApp/common/docker-compose.yml MyApp/development/docker-compose.yml
MyApp/production/docker-compose.yml
root@debian9:~# touch MyApp/production/content/Dockerfile MyApp/production/content/index.html
root@debian9:~# touch MyApp/development/content/Dockerfile MyApp/development/content/index.html
```

Utilisez l'utilitaire tree pour visualiser la structure du projet :

```
root@debian9:~# cd MyApp
root@debian9:~/MyApp# tree
.
├── common
│   └── docker-compose.yml
├── development
│   ├── content
│   │   ├── Dockerfile
│   │   └── index.html
│   └── docker-compose.yml
└── production
    ├── content
    │   ├── Dockerfile
    │   └── index.html
    └── docker-compose.yml

5 directories, 7 files
```

Ce projet comporte la structure suivante :

- **common**
  - La configuration commune aux deux environnements sera placée dans le fichier **MyApp/common/docker-compose.yml**.
- **development**
  - La configuration spécifique à l'environnement **development** sera placée dans le fichier **MyApp/development/docker-compose.yml**. Le contenu du fichier **index.html** sera **This is the development environement**.
- **production**
  - La configuration spécifique à l'environnement **production** sera placée dans le fichier **MyApp/production/docker-compose.yml**. Le contenu du fichier **index.html** sera **This is the production environment**.

Commencez par la création des deux Dockerfile pour **development** et **production**. Afin de garder l'exemple le plus simple que possible, ces deux fichiers sont identiques :

```
root@debian9:~/MyApp# vi development/content/Dockerfile

root@debian9:~/MyApp# cat development/content/Dockerfile
FROM tianon/true

VOLUME ["/usr/share/nginx/html/"]
ADD index.html /usr/share/nginx/html/

root@debian9:~/MyApp# cp development/content/Dockerfile production/content/Dockerfile
```

Créez maintenant le fichier **MyApp/common/docker-compose.yml** :

```
root@debian9:~/MyApp# vi common/docker-compose.yml

root@debian9:~/MyApp# cat common/docker-compose.yml
web:
  image: nginx
  ports:
    - 8082:80
```

Les deux fichiers **MyApp/development/docker-compose.yml** et **MyApp/production/docker-compose.yml** sont identiques :

```
root@debian9:~/MyApp# vi development/docker-compose.yml

root@debian9:~/MyApp# cat development/docker-compose.yml
web:
  extends:
    file: ../common/docker-compose.yml
    service: web
  volumes_from:
    - content

content:
  build: content

root@debian9:~/MyApp# cp development/docker-compose.yml production/docker-compose.yml
```

Éditez maintenant les deux fichiers index.html :

```
root@debian9:~/MyApp# vi development/content/index.html

root@debian9:~/MyApp# cat development/content/index.html
<html>
<body>
<center>This is the development environement</center>
</body>
</html>

root@debian9:~/MyApp# vi production/content/index.html

root@debian9:~/MyApp# cat production/content/index.html
<html>
<body>
<center>This is the production environement</center>
```

```
</body>
</html>
```

Placez-vous dans le sous-répertoire **development** et exécutez la commande **docker-compose up -d** :

```
root@debian9:~/MyApp/development# docker-compose up -d
Creating development_content_1...
Building content...
Step 1/3 : FROM tianon/true
 ---> 1298b2036003
Step 2/3 : VOLUME /usr/share/nginx/html/
 ---> Running in 8619de833add
 ---> 694e4f111996
Removing intermediate container 8619de833add
Step 3/3 : ADD index.html /usr/share/nginx/html/
 ---> f6fabac6703b
Removing intermediate container a9bec35dba66
Successfully built f6fabac6703b
Successfully tagged development_content:latest
Creating development_web_1...
```

La commande **docker-compose up** est une abréviation des commandes **docker-compose build && docker-compose run**. L'option **-d** a le même effet de son homologue de la commande **docker**.

Les options de la commande **docker-compose** sont :

```
root@debian9:~# docker-compose --help
Fast, isolated development environments using Docker.

Usage:
  docker-compose [options] [COMMAND] [ARGS...]
  docker-compose -h|--help

Options:
  --verbose                 Show more output
  --version                 Print version and exit
  -f, --file FILE           Specify an alternate compose file (default: docker-compose.yml)
  -p, --project-name NAME   Specify an alternate project name (default: directory name)

Commands:
  build     Build or rebuild services
  help      Get help on a command
  kill      Kill containers
  logs      View output from containers
  port      Print the public port for a port binding
  ps        List containers
  pull      Pulls service images
  rm        Remove stopped containers
  run       Run a one-off command
  scale     Set number of containers for a service
  start     Start services
  stop      Stop services
  restart   Restart services
  up        Create and start containers
```

Vérifiez que l'image **development_content** a été créée :

```
root@debian9:~/MyApp/development# docker images
REPOSITORY               TAG                 IMAGE ID            CREATED             SIZE
development_content      latest              f6fabac6703b        19 seconds ago      209B
...
```

Constatez la présence des deux conteneurs **nginx:latest** et **development_content** :

```
root@debian9:~/MyApp/development# docker ps -a
CONTAINER ID        IMAGE                 COMMAND                  CREATED             STATUS
PORTS                  NAMES
6955516dceff        nginx:latest          "nginx -g 'daemon ..."   28 seconds ago      Up 27 seconds
```

```
0.0.0.0:8082->80/tcp   development_web_1
9a1876d7a145        development_content   "/true"                  28 seconds ago      Exited (0) 27 seconds ago
development_content_1
...
```

Utilisez maintenant lynx pour consultez **http://localhost:8082** :

```
root@debian9:~/MyApp/development# lynx --dump http://localhost:8082
                    This is the development environement



root@debian9:~/MyApp/development#
```

Arrêtez docker-compose :

```
root@debian9:~/MyApp/development# docker-compose stop
Stopping development_web_1...
root@debian9:~/MyApp/development# docker ps -a
CONTAINER ID        IMAGE                 COMMAND                  CREATED             STATUS
PORTS               NAMES
6955516dceff        nginx:latest          "nginx -g 'daemon ..."   About a minute ago   Exited (0) 5 seconds ago
development_web_1
9a1876d7a145        development_content   "/true"                  About a minute ago   Exited (0) About a minute
ago                        development_content_1
...
```

Placez-vous maintenant dans le sous-répertoire **production** et exécutez de nouveau la commande **docker-compose up -d** :

```
root@debian9:~/MyApp/development# cd ../production/
root@debian9:~/MyApp/production# docker-compose up -d
Creating production_content_1...
Building content...
Step 1/3 : FROM tianon/true
 ---> 1298b2036003
```

```
Step 2/3 : VOLUME /usr/share/nginx/html/
 ---> Using cache
 ---> 694e4f111996
Step 3/3 : ADD index.html /usr/share/nginx/html/
 ---> 61bcd73aff6e
Removing intermediate container 18af8bcb48ce
Successfully built 61bcd73aff6e
Successfully tagged production_content:latest
Creating production_web_1...
```

Notez la création du conteneur **production_content** :

```
root@debian9:~/MyApp/production# docker ps -a
CONTAINER ID        IMAGE                COMMAND                  CREATED             STATUS
PORTS                  NAMES
8a2c7346a5db        nginx:latest         "nginx -g 'daemon ..."   4 seconds ago       Up 3 seconds
0.0.0.0:8082->80/tcp   production_web_1
5fde5e7cbd47        production_content    "/true"                  4 seconds ago       Exited (0) 3 seconds ago
production_content_1
6955516dceff        nginx:latest         "nginx -g 'daemon ..."   2 minutes ago       Exited (0) About a minute
ago                          development_web_1
9a1876d7a145        development_content   "/true"                  2 minutes ago       Exited (0) 2 minutes ago
development_content_1
...
```

En étant dans le contexte **production**, il est possible d'utiliser la commande **docker-compose ps** :

```
root@debian9:~/MyApp/production# docker-compose ps
        Name                   Command          State           Ports
---------------------------------------------------------------------------
production_content_1   /true                    Exit 0
production_web_1       nginx -g daemon off;   Up      0.0.0.0:8082->80/tcp
```

De même en utilisant la même commande dans le répertoire **development**, on peut constater l'état de l'environnement **development** :

```
root@debian9:~/MyApp/production# cd ../development/
root@debian9:~/MyApp/development# docker-compose ps
        Name                  Command          State    Ports
-------------------------------------------------------------
development_content_1   /true                  Exit 0
development_web_1       nginx -g daemon off;   Exit 0
```

Utilisez maintenant lynx pour consultez **http://localhost:8082** :

```
root@debian9:~/MyApp/development# lynx --dump http://localhost:8082
                     This is the production environement



root@debian9:~/MyApp/development#
```

Installer Wordpress avec Docker Compose

Créez maintenant le répertoire **wordpress1** dans /root :

```
root@debian9:~/MyApp/development# cd ~
root@debian9:~# mkdir wordpress1
```

Placez-vous dans le répertoire et créer le fichier **docker-compose.yaml**

```
root@debian9:~# cd wordpress1
root@debian9:~/wordpress1# vi docker-compose.yaml
root@debian9:~/wordpress1# cat docker-compose.yaml
version: "3.3"
services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: fenestros
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
volumes:
  db_data: {}
```

Exécutez la commande docker-compose :

```
root@debian9:~/wordpress1# docker-compose up -d
```

Vérifiez que le Wordpress fonctionne :

```
root@debian9:~/wordpress1# lynx --dump http://10.0.2.60:8000
   WordPress
   Select a default language [English (United States)________]

   Continue
root@debian9:~# docker ps -a
```

```
CONTAINER ID        IMAGE                 COMMAND                  CREATED             STATUS
PORTS                  NAMES
29afa2a7fdb5        wordpress:latest      "docker-entrypoint.s…"   21 minutes ago      Up 20 minutes
0.0.0.0:8000->80/tcp   wordpress1_wordpress_1
...

root@debian9:~/wordpress1# docker inspect wordpress1_wordpress_1 | grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "172.17.0.7",
                    "IPAddress": "172.17.0.7",
root@debian9:~/wordpress1# lynx --dump http://172.17.0.7
   WordPress
   Select a default language [English (United States)________]

   Continue
```

LAB #2 - Docker Machine

2.1 - Présentation

Docker Machine est un outil qui vous permet d'installer docker sur des hôtes virtuels et de gérer les hôtes en utilisant des commandes spécifiques à docker-machine. Il est donc possible d'utiliser cet outil pour créer des hôtes docker localement, sur le réseau, dans un data center ou dans le cloud (Azure, AWS, Digital Ocean par exemple).

Le jeu de commandes de docker-machine permet de démarrer, surveiller, arrêter et re-démarrer un hôte géré, de mettre à jour le client/daemon docker et de configurer un client docker afin qu'il “parle” à votre machine hôte.

Pour installer docker-machine sur votre VM **debian9**, utilisez la commande suivante :

```
root@debian9:~# curl -L https://github.com/docker/machine/releases/download/v0.12.2/docker-machine-`uname -s`-
`uname -m` >/tmp/docker-machine && chmod +x /tmp/docker-machine && cp /tmp/docker-machine /usr/local/bin/docker-
machine
```

2.2 - Création de Machines Virtuelles Docker

La création d'une machine se fait simplement en utilisant la commande **docker-machine** avec la sous-commande **create**. Cette sous-commande prend l'option **-driver** ou **-d** qui indique le fournisseur à utiliser :

Fournisseur	**driver**
Amazon Web Services	amazonec2
Digital Ocean	digitalocean
Exoscale	exoscale
Google Compute Engine	google
IBM Softlayer	softlayer
Microsoft Hyper-V	hyperv
Microsoft Azure	azure
OpenStack	openstack
Oracle VirtualBox	virtualbox
Rackspace	rackspace
VMware Fusion	vmwarefusion
VMware vCloud Air	vmwarevcloudair
VMware vSphere	vmwarevsphere

Commencez par installer Oracle VirtualBox:

```
root@debian9:~/wordpress# cd ~
root@debian9:~# apt install virtualbox-6.0
```

Créez maintenant la machine virtuelle **manager1** :

```
root@debian9:~# docker-machine create --driver virtualbox manager1
Creating CA: /root/.docker/machine/certs/ca.pem
Creating client certificate: /root/.docker/machine/certs/cert.pem
Running pre-create checks...
(manager1) Image cache directory does not exist, creating it at /root/.docker/machine/cache...
```

```
(manager1) No default Boot2Docker ISO found locally, downloading the latest release...
(manager1) Latest release for github.com/boot2docker/boot2docker is v17.06.2-ce
(manager1) Downloading /root/.docker/machine/cache/boot2docker.iso from
https://github.com/boot2docker/boot2docker/releases/download/v17.06.2-ce/boot2docker.iso...
(manager1) 0%....10%....20%....30%....40%....50%....60%....70%....80%....90%....100%
Creating machine...
(manager1) Copying /root/.docker/machine/cache/boot2docker.iso to
/root/.docker/machine/machines/manager1/boot2docker.iso...
(manager1) Creating VirtualBox VM...
(manager1) Creating SSH key...
(manager1) Starting the VM...
(manager1) Check network to re-create if needed...
(manager1) Found a new host-only adapter: "vboxnet0"
(manager1) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: docker-
machine env manager1
```

Les options de la commande **docker-machine** sont :

```
root@debian9:~# docker-machine --help
Usage: docker-machine [OPTIONS] COMMAND [arg...]

Create and manage machines running Docker.

```

```
Version: 0.12.2, build 9371605

Author:
  Docker Machine Contributors - <https://github.com/docker/machine>

Options:
  --debug, -D                   Enable debug mode
  --storage-path, -s "/root/.docker/machine"  Configures storage path [$MACHINE_STORAGE_PATH]
  --tls-ca-cert                 CA to verify remotes against [$MACHINE_TLS_CA_CERT]
  --tls-ca-key                  Private key to generate certificates [$MACHINE_TLS_CA_KEY]
  --tls-client-cert                 Client cert to use for TLS [$MACHINE_TLS_CLIENT_CERT]
  --tls-client-key              Private key used in client TLS auth [$MACHINE_TLS_CLIENT_KEY]
  --github-api-token                Token to use for requests to the Github API [$MACHINE_GITHUB_API_TOKEN]
  --native-ssh                  Use the native (Go-based) SSH implementation. [$MACHINE_NATIVE_SSH]
  --bugsnag-api-token               BugSnag API token for crash reporting [$MACHINE_BUGSNAG_API_TOKEN]
  --help, -h                    show help
  --version, -v                 print the version
Commands:
  active        Print which machine is active
  config        Print the connection config for machine
  create        Create a machine
  env           Display the commands to set up the environment for the Docker client
  inspect       Inspect information about a machine
  ip            Get the IP address of a machine
  kill          Kill a machine
  ls            List machines
  provision     Re-provision existing machines
  regenerate-certs  Regenerate TLS Certificates for a machine
  restart       Restart a machine
  rm            Remove a machine
  ssh           Log into or run a command on a machine with SSH.
  scp           Copy files between machines
  start         Start a machine
  status        Get the status of a machine
  stop           Stop a machine
  upgrade        Upgrade a machine to the latest version of Docker
  url            Get the URL of a machine
  version        Show the Docker Machine version or a machine docker version
  help           Shows a list of commands or help for one command
Run 'docker-machine COMMAND --help' for more information on a command.
```

Créez maintenant 5 travailleurs - **worker1** jusqu'à **worker5** :

```
root@debian9:~# docker-machine create --driver virtualbox worker1
Running pre-create checks...
Creating machine...
(worker1) Copying /root/.docker/machine/cache/boot2docker.iso to
/root/.docker/machine/machines/worker1/boot2docker.iso...
(worker1) Creating VirtualBox VM...
(worker1) Creating SSH key...
(worker1) Starting the VM...
(worker1) Check network to re-create if needed...
(worker1) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: docker-
machine env worker1
```

```
root@debian9:~# docker-machine create --driver virtualbox worker2
...
```

```
root@debian9:~# docker-machine create --driver virtualbox worker3
...
root@debian9:~# docker-machine create --driver virtualbox worker4
...
root@debian9:~# docker-machine create --driver virtualbox worker5
...
```

Les options de la sous-commande **create** de la commande **docker-machine** sont :

```
root@debian9:~# docker-machine create --help
Usage: docker-machine create [OPTIONS] [arg...]

Create a machine

Description:
   Run 'docker-machine create --driver name' to include the create flags for that driver in the help text.

Options:
   --driver, -d "virtualbox"                                          Driver to create machine with.
[$MACHINE_DRIVER]
   --engine-env [--engine-env option --engine-env option]                       Specify environment variables to
set in the engine
   --engine-insecure-registry [--engine-insecure-registry option --engine-insecure-registry option] Specify
insecure registries to allow with the created engine
   --engine-install-url "https://get.docker.com"                          Custom URL to use for engine
installation [$MACHINE_DOCKER_INSTALL_URL]
   --engine-label [--engine-label option --engine-label option]                   Specify labels for the
created engine
   --engine-opt [--engine-opt option --engine-opt option]                       Specify arbitrary flags to
include with the created engine in the form flag=value
   --engine-registry-mirror [--engine-registry-mirror option --engine-registry-mirror option]       Specify
registry mirrors to use [$ENGINE_REGISTRY_MIRROR]
   --engine-storage-driver                                          Specify a storage driver to use with the engine
   --swarm                                              Configure Machine to join a Swarm cluster
   --swarm-addr                                             addr to advertise for Swarm (default: detect and use
the machine IP)
   --swarm-discovery                                            Discovery service to use with Swarm
   --swarm-experimental                                         Enable Swarm experimental features
   --swarm-host "tcp://0.0.0.0:3376"                                  ip/socket to listen on for Swarm master
   --swarm-image "swarm:latest"                                       Specify Docker image to use for Swarm
[$MACHINE_SWARM_IMAGE]
   --swarm-join-opt [--swarm-join-opt option --swarm-join-opt option]                   Define arbitrary flags
for Swarm join
   --swarm-master                                           Configure Machine to be a Swarm master
   --swarm-opt [--swarm-opt option --swarm-opt option]                              Define arbitrary flags for Swarm
master
   --swarm-strategy "spread"                                      Define a default scheduling strategy for Swarm
   --tls-san [--tls-san option --tls-san option]                            Support extra SANs for TLS certs
   --virtualbox-boot2docker-url                                     The URL of the boot2docker image. Defaults to
the latest available version [$VIRTUALBOX_BOOT2DOCKER_URL]
   --virtualbox-cpu-count "1"                                     number of CPUs for the machine (-1 to use the
number of CPUs available) [$VIRTUALBOX_CPU_COUNT]
   --virtualbox-disk-size "20000"                                 Size of disk for host in MB
[$VIRTUALBOX_DISK_SIZE]
   --virtualbox-host-dns-resolver                                   Use the host DNS resolver
[$VIRTUALBOX_HOST_DNS_RESOLVER]
   --virtualbox-hostonly-cidr "192.168.99.1/24"                               Specify the Host Only CIDR
[$VIRTUALBOX_HOSTONLY_CIDR]
   --virtualbox-hostonly-nicpromisc "deny"                                Specify the Host Only Network Adapter
Promiscuous Mode [$VIRTUALBOX_HOSTONLY_NIC_PROMISC]
   --virtualbox-hostonly-nictype "82540EM"                                Specify the Host Only Network Adapter
Type [$VIRTUALBOX_HOSTONLY_NIC_TYPE]
   --virtualbox-hostonly-no-dhcp                                    Disable the Host Only DHCP Server
[$VIRTUALBOX_HOSTONLY_NO_DHCP]
   --virtualbox-import-boot2docker-vm                                   The name of a Boot2Docker VM to import
[$VIRTUALBOX_BOOT2DOCKER_IMPORT_VM]
   --virtualbox-memory "1024"                                     Size of memory for host in MB
[$VIRTUALBOX_MEMORY_SIZE]
```

```
   --virtualbox-nat-nictype "82540EM"                                Specify the Network Adapter Type
[$VIRTUALBOX_NAT_NICTYPE]
   --virtualbox-no-dns-proxy                                        Disable proxying all DNS requests to the host
[$VIRTUALBOX_NO_DNS_PROXY]
   --virtualbox-no-share                                           Disable the mount of your home directory
[$VIRTUALBOX_NO_SHARE]
   --virtualbox-no-vtx-check                                        Disable checking for the availability of
hardware virtualization before the vm is started [$VIRTUALBOX_NO_VTX_CHECK]
   --virtualbox-share-folder                                        Mount the specified directory instead of the
default home location. Format: dir:name [$VIRTUALBOX_SHARE_FOLDER]
   --virtualbox-ui-type "headless"                                   Specify the UI Type:
(gui|sdl|headless|separate) [$VIRTUALBOX_UI_TYPE]
```

2.3 - Lister les VM Docker

Pour lister les VM Docker ainsi que leurs états, il convient d'utilise la sous-commande **ls** de la commande **docker-machine** :

```
root@debian9:~# docker-machine ls
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER        ERRORS
manager1   -        virtualbox   Running   tcp://192.168.99.100:2376           v17.06.2-ce
worker1    -        virtualbox   Running   tcp://192.168.99.101:2376           v17.06.2-ce
worker2    -        virtualbox   Running   tcp://192.168.99.102:2376           v17.06.2-ce
worker3    -        virtualbox   Running   tcp://192.168.99.103:2376           v17.06.2-ce
worker4    -        virtualbox   Running   tcp://192.168.99.104:2376           v17.06.2-ce
worker5    -        virtualbox   Running   tcp://192.168.99.105:2376           v17.06.2-ce
```

2.4 - Obtenir l'adresse IP des VM

Une autre façon d'obtenir les adresses IP des VM est d'utiliser la sous-commande **ip** :

```
root@debian9:~# docker-machine ip manager1
```

```
192.168.99.100
root@debian9:~# docker-machine ip worker1
192.168.99.101
root@debian9:~# docker-machine ip worker2
192.168.99.102
root@debian9:~# docker-machine ip worker3
192.168.99.103
root@debian9:~# docker-machine ip worker4
192.168.99.104
root@debian9:~# docker-machine ip worker5
192.168.99.105
```

2.5 - Se connecter à une VM Docker

Pour se connecter à une VM Docker, il convient d'utiliser la sous-commande **ssh** de la commande **docker-machine** :

```
root@debian9:~# docker-machine ssh manager1
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           \______ o           __/
             \    \         __/
              \____\_______/
 _                 _   ____     _            _
| |__   ___   ___ | |_|___ \ __| | ___   ___| | _____ _ __
| '_ \ / _ \ / _ \| __| __) / _` |/ _ \ / __| |/ / _ \ '__|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ \___/ \___/ \__|_____\__,_|\___/ \___|_|\_\___|_|
Boot2Docker version 17.06.2-ce, build HEAD : ff16afa - Wed Sep  6 00:17:25 UTC 2017
Docker version 17.06.2-ce, build cec0b72
```

```
docker@manager1:~$ exit
```

Important - Notez que la distribution de la VM est **Boot2Docker**. Cette distribution est basée sur **Tiny Core Linux**, s'exécute entièrement dans la mémoire vive, pèse 27 Mo et démarre en approximativement 5 secondes.

Installez maintenant le paquet **mlocate** :

```
root@debian9:~# apt install mlocate
...
```

Ayant été créées par root, les VM Docker ainsi que leurs fichiers associés sont stockés dans le répertoire **/root/.docker/machine/machines/** :

```
root@debian9:~# updatedb
root@debian9:~# locate manager1
/root/.docker/machine/machines/manager1
/root/.docker/machine/machines/manager1/boot2docker.iso
/root/.docker/machine/machines/manager1/ca.pem
/root/.docker/machine/machines/manager1/cert.pem
/root/.docker/machine/machines/manager1/config.json
/root/.docker/machine/machines/manager1/disk.vmdk
/root/.docker/machine/machines/manager1/id_rsa
/root/.docker/machine/machines/manager1/id_rsa.pub
/root/.docker/machine/machines/manager1/key.pem
/root/.docker/machine/machines/manager1/manager1
/root/.docker/machine/machines/manager1/server-key.pem
/root/.docker/machine/machines/manager1/server.pem
/root/.docker/machine/machines/manager1/manager1/Logs
/root/.docker/machine/machines/manager1/manager1/manager1.vbox
/root/.docker/machine/machines/manager1/manager1/manager1.vbox-prev
/root/.docker/machine/machines/manager1/manager1/Logs/VBox.log
```

LAB #3 - Docker Swarm

3.1 - Présentation

Docker Swarm est un utilitaire qui permet de gérer un cluster pour déployer des conteneurs en permettant une imitation du comportement de docker sur une seule machine.

3.2 - Initialiser Docker Swarm

Pour initialiser Docker swarm, il convient d'utiliser la commande **docker swarm init** à partir de la VM Docker **manager1** en stipulant l'adresse IP de manager1 :

```
root@debian9:~# docker-machine ssh manager1
docker@manager1:~$ docker swarm init --advertise-addr 192.168.99.100
Swarm initialized: current node (yuwpmvtfmdxn8i7nllkyzkxkp) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token
SWMTKN-1-5bd9w9tapfqmd41f2psqdkoqwfo48fqsznnalk2slc28vlp6uh-004kp8y71m09nd7p8ft7ldku0 192.168.99.100:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Notez que les ports suivants doivent être ouverts sur un nœud manager : 22/tcp, 2376/tcp, 2377/tcp, 7946/tcp, 7946/udp et 4789/udp.

3.3 - Le Statut Leader

Consultez le statut de la VM Docker manager1 :

```
docker@manager1:~$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
yuwpmvtfmdxn8i7nllkyzkxkp *   manager1            Ready               Active              Leader
```

A un instant t il ne peut y avoir q'un seul **Leader**. Il est possible de créer d'autres nœuds de gestion en le rejoignant à swarm en utilisant le token prévu à cet effet. Par contre ces nœuds de gestion restent en attente d'une éventuelle défaillance du Leader actuel.

Pour connaître le token nécessaire pour rejoindre swarm en tant que nœud de gestion, saisissez la commande suivante :

```
docker@manager1:~$ docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join --token
SWMTKN-1-5bd9w9tapfqmd41f2psqdkoqwfo48fqsznnalk2slc28vlp6uh-8rvbxvqewsrv6yyts7z2lq9pt 192.168.99.100:2377
```

3.4 - Rejoindre le Swarm

Rejoignez les 5 machines travailleurs à swarm en utilisant le token **worker** :

```
docker@worker1:~$ docker swarm join --token
SWMTKN-1-5bd9w9tapfqmd41f2psqdkoqwfo48fqsznnalk2slc28vlp6uh-004kp8y71m09nd7p8ft7ldku0 192.168.99.100:2377
This node joined a swarm as a worker.
```

```
docker@worker2:~$ docker swarm join --token
SWMTKN-1-5bd9w9tapfqmd41f2psqdkoqwfo48fqsznnalk2slc28vlp6uh-004kp8y71m09nd7p8ft7ldku0 192.168.99.100:2377
This node joined a swarm as a worker.
```

```
docker@worker3:~$ docker swarm join --token
SWMTKN-1-5bd9w9tapfqmd41f2psqdkoqwfo48fqsznnalk2slc28vlp6uh-004kp8y71m09nd7p8ft7ldku0 192.168.99.100:2377
This node joined a swarm as a worker.
```

```
docker@worker4:~$ docker swarm join --token
```

```
SWMTKN-1-5bd9w9tapfqmd41f2psqdkoqwfo48fqsznnalk2slc28vlp6uh-004kp8y71m09nd7p8ft7ldku0 192.168.99.100:2377
This node joined a swarm as a worker.
```

```
docker@worker5:~$ docker swarm join --token
SWMTKN-1-5bd9w9tapfqmd41f2psqdkoqwfo48fqsznnalk2slc28vlp6uh-004kp8y71m09nd7p8ft7ldku0 192.168.99.100:2377
This node joined a swarm as a worker.
```

Notez que les ports suivants doivent être ouverts sur un nœud worker : 22/tcp, 2376/tcp, 7946/tcp, 7946/udp et 4789/udp.

L'état des VM Docker peut être consulter en utilisant de nouveau la commande **docker node ls** :

```
docker@manager1:~$ docker node ls
ID                           HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
1f5qtolgtonqmhjk5ppwc8x1b     worker1             Ready               Active
kmyjdwp9ojhzje4hlw7ffhuxv     worker2             Ready               Active
oyszb44k8yw5btz3c1wq2ot2e     worker4             Ready               Active
p6jpyopzzy0zg4znegi63hzjq     worker5             Ready               Active
yitkfnk99ecisrny9g3r9kfhk     worker3             Ready               Active
yuwpmvtfmdxn8i7nllkyzkxkp *   manager1            Ready               Active              Leader
```

Notez que vous ne pouvez pas utiliser cette commande à partir d'un travailleur :

```
docker@worker5:~$ docker node ls
Error response from daemon: This node is not a swarm manager. Worker nodes can't be used to view or modify
cluster state. Please run this command on a manager node or promote the current node to a manager.
```

3.5 - Consulter les Informations de Swarm

Il est possible de visualiser les informations concernant le swarm en utilisant la commande **docker info** :

```
docker@manager1:~$ docker info
...
```

```
Swarm: active
 NodeID: yuwpmvtfmdxn8i7nllkyzkxkp
 Is Manager: true
 ClusterID: sqll9xmii9qkrd35d1limn1od
 Managers: 1
 Nodes: 6
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
  Heartbeat Tick: 1
  Election Tick: 3
 Dispatcher:
  Heartbeat Period: 5 seconds
 CA Configuration:
  Expiry Duration: 3 months
  Force Rotate: 0
 Root Rotation In Progress: false
 Node Address: 192.168.99.100
 Manager Addresses:
  192.168.99.100:2377
...
```

Important - Quand le moteur Docker est en mode swarm, les noeuds de gestion implémentent le **Raft Consensus Algorithm** pour gérer l'état du cluster.

3.6 - Démarrer un Service

Dans cet exemple, nous allons démarrer le service **nginx** avec les propriétés suivantes :

* Mappage du port nginx sur le port 80 de la machine hôte,
* 5 instances du service,
* Un nom unique de **web**.

```
docker@manager1:~$ docker service create --replicas 5 -p 80:80 --name web nginx
4xtuwgbvr17lvfzoumh1y4mq4
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
```

Pour consulter l'état de ce service, utilisez la commande **docker service ls** :

```
docker@manager1:~$ docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
4xtuwgbvr17l        web                 replicated          5/5                 nginx:latest        *:80->80/tcp
```

Ce service fonctionne dans des conteneurs Docker :

```
docker@manager1:~$ docker service ps web
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE
ERROR               PORTS
jkm2hapcthht        web.1               nginx:latest        worker3             Running             Running about
a minute ago
q55eqdhr1qf1        web.2               nginx:latest        worker4             Running             Running about
a minute ago
imqdkw4ei6gs        web.3               nginx:latest        manager1            Running             Running about
a minute ago
k4vjd0g7ijww        web.4               nginx:latest        worker1             Running             Running about
a minute ago
b7xbmy1npgf9        web.5               nginx:latest        worker2             Running             Running about
a minute ago
```

Important - Notez qu'il n'y a pas de conteneur sur worker5.

Pour constater le lancement du daemon nginx, lancez la commande **docker ps** sur la machine **manager1** :

```
docker@manager1:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
NAMES
4107cb687eda        nginx:latest        "nginx -g 'daemon ..."   2 minutes ago       Up 2 minutes        80/tcp
web.3.imqdkw4ei6gskwacnb4pime5f
```

Connectez-vous sur chaque VM Docker pour constater que le service nginx fonctionne :

```
docker@manager1:/$ curl 192.168.99.100
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
```

```
</body>
</html>
docker@manager1:/$ curl 192.168.99.101
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
docker@manager1:/$ curl 192.168.99.102
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
```

```
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
docker@manager1:/$ curl 192.168.99.103
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
```

```
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
docker@manager1:/$ curl 192.168.99.104
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>
```

```
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
docker@manager1:/$ curl 192.168.99.105
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Important - Notez que le service est même disponible en consultant l'adresse IP de worker5.

3.7 - Augmentation et Réduction du Service

Actuellement, il existe 5 conteneurs en cours d'exécution. Pour procéder à un scale-up à 8 conteneurs, il convient d'utiliser la commande **docker service scale** :

```
docker@manager1:/$ docker service scale web=8
web scaled to 8
```

Notez que la commande **docker service ls** confirme le fait qu'il y a 8 replicas :

```
docker@manager1:/$ docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
4xtuwgbvr17l        web                 replicated          8/8                 nginx:latest        *:80->80/tcp
```

Des trois replicas supplémentaires, deux ont été lancés sur worker5 tandis que le troisième a été lancé sur worker1 :

```
docker@manager1:/$ docker service ps web
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE
ERROR               PORTS
jkm2hapcthht        web.1               nginx:latest        worker3             Running             Running 20
minutes ago
q55eqdhr1qf1        web.2               nginx:latest        worker4             Running             Running 20
minutes ago
imqdkw4ei6gs        web.3               nginx:latest        manager1            Running             Running 20
minutes ago
k4vjd0g7ijww        web.4               nginx:latest        worker1             Running             Running 20
minutes ago
b7xbmy1npgf9        web.5               nginx:latest        worker2             Running             Running 20
minutes ago
kg3bivcg0wln        web.6               nginx:latest        worker5             Running             Running 47
seconds ago
ik3u0jfgey64        web.7               nginx:latest        worker5             Running             Running 47
seconds ago
```

```
6bw5ptw7xao8        web.8               nginx:latest        worker1             Running             Running 57
seconds ago
```

3.8 - Consulter le Statut d'un Noeud

Pour se renseigner sur le statut du nœud courant, il convient d'utiliser la commande **docker node inspect** avec le mot clef **self** :

```
docker@manager1:/$ docker node inspect self
[
    {
        "ID": "yuwpmvtfmdxn8i7nllkyzkxkp",
        "Version": {
            "Index": 9
        },
        "CreatedAt": "2017-09-08T11:43:55.289178512Z",
        "UpdatedAt": "2017-09-08T11:43:55.89870884Z",
        "Spec": {
            "Labels": {},
            "Role": "manager",
            "Availability": "active"
        },
        "Description": {
            "Hostname": "manager1",
            "Platform": {
                "Architecture": "x86_64",
                "OS": "linux"
            },
            "Resources": {
                "NanoCPUs": 1000000000,
                "MemoryBytes": 1044123648
            },
            "Engine": {
                "EngineVersion": "17.06.2-ce",
                "Labels": {
                    "provider": "virtualbox"
                },
                "Plugins": [
                    {
                        "Type": "Log",
                        "Name": "awslogs"
                    },
                    {
                        "Type": "Log",
                        "Name": "fluentd"
                    },
                    {
                        "Type": "Log",
                        "Name": "gcplogs"
                    },
                    {
                        "Type": "Log",
                        "Name": "gelf"
                    },
                    {
                        "Type": "Log",
                        "Name": "journald"
                    },
                    {
                        "Type": "Log",
                        "Name": "json-file"
                    },
                    {
                        "Type": "Log",
                        "Name": "logentries"
                    },
                    {
                        "Type": "Log",
                        "Name": "splunk"
                    },
                    {
                        "Type": "Log",
                        "Name": "syslog"
                    },
                    {
                        "Type": "Network",
                        "Name": "bridge"
                    },
                    {
                        "Type": "Network",
                        "Name": "host"
                    },
                    {
                        "Type": "Network",
                        "Name": "macvlan"
                    },
                    {
                        "Type": "Network",
                        "Name": "null"
                    },
                    {
                        "Type": "Network",
                        "Name": "overlay"
                    },
                    {
                        "Type": "Volume",
                        "Name": "local"
                    }
                ]
            },
            "TLSInfo": {
                "TrustRoot": "-----BEGIN CERTIFICATE-----
```

```
\nMIIBajCCARCgAwIBAgIUNuU4I89kxId2QXulofRKxJa9XRcwCgYIKoZIzj0EAwIw\nEzERMA8GA1UEAxMIc3dhcm0tY2EwHhcNMTcwOTA4MTEzO
TAwWhcNMzcwOTAzMTEz\nOTAwWjATMREwDwYDVQQDEwhzd2FybS1jYTBZMBMGByqGSM49AgEGCCqGSM49AwEH\nA0IABEqgLUbyjyNuP35aAzW+aq
VB8AkghvpF5hq1KnMveHbl4Ilr+EyDjlYZkbnt\nGb/xmsy/tOP8uz598ZX/JlR4fZyjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMB\nAf8EBTA
DAQH/MB0GA1UdDgQWBBSzoKGrN0ELfEIMsjxuYj5LAckD2jAKBggqhkjO\nPQQDAgNIADBFAiB34DOvDtIYjJ+GzbPMGu9Dd/cJGvy7CJg1tNUG3S
oOrAIhAJZ4\nTJBucTomFSDsj5Y/R6TfhcpXpsksk7JwYgEglu44\n-----END CERTIFICATE-----\n",
                "CertIssuerSubject": "MBMxETAPBgNVBAMTCHN3YXJtLWNh",
                "CertIssuerPublicKey":
"MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAESqAtRvKPI24/floDNb5qpUHwCSCG+kXmGrUqcy94duXgiWv4TIOOVhmRue0Zv/GazL+04/y7Pn3x
lf8mVHh9nA=="
            }
        },
        "Status": {
            "State": "ready",
            "Addr": "192.168.99.100"
        },
        "ManagerStatus": {
            "Leader": true,
            "Reachability": "reachable",
            "Addr": "192.168.99.100:2377"
        }
    }
]
```

Pour se renseigner sur le statut d'un autre nœud, il convient d'utiliser la commande **docker node inspect** avec le nom du nœud concerné :

```
docker@manager1:/$ docker node inspect worker1
[
    {
        "ID": "1f5qtolgtonqmhjk5ppwc8x1b",
        "Version": {
            "Index": 15
        },
        "CreatedAt": "2017-09-08T11:48:42.011596185Z",
        "UpdatedAt": "2017-09-08T11:48:42.093455479Z",
        "Spec": {
            "Labels": {},
            "Role": "worker",
            "Availability": "active"
        },
        "Description": {
            "Hostname": "worker1",
            "Platform": {
                "Architecture": "x86_64",
                "OS": "linux"
            },
            "Resources": {
                "NanoCPUs": 1000000000,
                "MemoryBytes": 1044123648
            },
            "Engine": {
                "EngineVersion": "17.06.2-ce",
                "Labels": {
                    "provider": "virtualbox"
                },
                "Plugins": [
                    {
                        "Type": "Log",
                        "Name": "awslogs"
                    },
                    {
                        "Type": "Log",
                        "Name": "fluentd"
                    },
                    {
                        "Type": "Log",
                        "Name": "gcplogs"
                    },
                    {
                        "Type": "Log",
                        "Name": "gelf"
                    },
                    {
                        "Type": "Log",
                        "Name": "journald"
                    },
                    {
                        "Type": "Log",
                        "Name": "json-file"
                    },
                    {
                        "Type": "Log",
                        "Name": "logentries"
                    },
                    {
                        "Type": "Log",
                        "Name": "splunk"
                    },
                    {
                        "Type": "Log",
                        "Name": "syslog"
                    },
                    {
                        "Type": "Network",
                        "Name": "bridge"
                    },
                    {
                        "Type": "Network",
                        "Name": "host"
                    },
                    {
                        "Type": "Network",
                        "Name": "macvlan"
                    },
                    {
                        "Type": "Network",
                        "Name": "null"
                    },
                    {
                        "Type": "Network",
                        "Name": "overlay"
                    },
                    {
                        "Type": "Volume",
                        "Name": "local"
                    }
                ]
            },
            "TLSInfo": {
                "TrustRoot": "-----BEGIN CERTIFICATE-----
\nMIIBajCCARCgAwIBAgIUNuU4I89kxId2QXulofRKxJa9XRcwCgYIKoZIzj0EAwIw\nEzERMA8GA1UEAxMIc3dhcm0tY2EwHhcNMTcwOTA4MTEzO
TAwWhcNMzcwOTAzMTEz\nOTAwWjATMREwDwYDVQQDEwhzd2FybS1jYTBZMBMGByqGSM49AgEGCCqGSM49AwEH\nA0IABEqgLUbyjyNuP35aAzW+aq
VB8AkghvpF5hq1KnMveHbl4Ilr+EyDjlYZkbnt\nGb/xmsy/tOP8uz598ZX/JlR4fZyjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMB\nAf8EBTA
DAQH/MB0GA1UdDgQWBBSzoKGrN0ELfEIMsjxuYj5LAckD2jAKBggqhkjO\nPQQDAgNIADBFAiB34DOvDtIYjJ+GzbPMGu9Dd/cJGvy7CJg1tNUG3S
oOrAIhAJZ4\nTJBucTomFSDsj5Y/R6TfhcpXpsksk7JwYgEglu44\n-----END CERTIFICATE-----\n",
                "CertIssuerSubject": "MBMxETAPBgNVBAMTCHN3YXJtLWNh",
                "CertIssuerPublicKey":
"MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAESqAtRvKPI24/floDNb5qpUHwCSCG+kXmGrUqcy94duXgiWv4TIOOVhmRue0Zv/GazL+04/y7Pn3x
lf8mVHh9nA=="
            }
        },
        "Status": {
            "State": "ready",
            "Addr": "192.168.99.101"
        }
    }
]
```

L'option **-pretty** produit une sortie plus facilement lisible :

```
docker@manager1:/$ docker node inspect --pretty worker1
ID:                     1f5qtolgtonqmhjk5ppwc8x1b
Hostname:               worker1
Joined at:              2017-09-08 11:48:42.011596185 +0000 utc
Status:
 State:                 Ready
 Availability:          Active
 Address:               192.168.99.101
Platform:
 Operating System:      linux
 Architecture:          x86_64
Resources:
 CPUs:                  1
 Memory:                995.8MiB
Plugins:
 Log:               awslogs, fluentd, gcplogs, gelf, journald, json-file, logentries, splunk, syslog
 Network:               bridge, host, macvlan, null, overlay
 Volume:                local
Engine Version:         17.06.2-ce
Engine Labels:
 - provider=virtualbox
TLS Info:
 TrustRoot:
-----BEGIN CERTIFICATE-----
MIIBajCCARCgAwIBAgIUNuU4I89kxId2QXulofRKxJa9XRcwCgYIKoZIzj0EAwIw
EzERMA8GA1UEAxMIc3dhcm0tY2EwHhcNMTcwOTA4MTEzOTAwWhcNMzcwOTAzMTEz
OTAwWjATMREwDwYDVQQDEwhzd2FybS1jYTBZMBMGByqGSM49AgEGCCqGSM49AwEH
A0IABEqgLUbyjyNuP35aAzW+aqVB8AkghvpF5hq1KnMveHbl4Ilr+EyDjlYZkbnt
Gb/xmsy/tOP8uz598ZX/JlR4fZyjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMB
Af8EBTADAQH/MB0GA1UdDgQWBBSzoKGrN0ELfEIMsjxuYj5LAckD2jAKBggqhkjO
PQQDAgNIADBFAiB34DOvDtIYjJ+GzbPMGu9Dd/cJGvy7CJg1tNUG3SoOrAIhAJZ4
TJBucTomFSDsj5Y/R6TfhcpXpsksk7JwYgEglu44
```

```
-----END CERTIFICATE-----

 Issuer Subject:    MBMxETAPBgNVBAMTCHN3YXJtLWNh
 Issuer Public Key:
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAESqAtRvKPI24/floDNb5qpUHwCSCG+kXmGrUqcy94duXgiWv4TIOOVhmRue0Zv/GazL+04/y7Pn3xl
f8mVHh9nA==
```

3.9 - Haute Disponibilité

Quand un nœud est actif, il est capable de recevoir de nouvelles tâches à partir du manager :

- pendant un scale-up,
- pendant une une mise à jour progressive,
- quand un autre nœud reçois une instruction de se mettre en indisponibilité,
- quand un service se mets en échec sur un autre nœud

Rappelez-vous que la swarm contient 6 VM Docker :

```
docker@manager1:/$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
1f5qtolgtonqmhjk5ppwc8x1b     worker1             Ready               Active              
kmyjdwp9ojhzje4hlw7ffhuxv     worker2             Ready               Active              
oyszb44k8yw5btz3c1wq2ot2e     worker4             Ready               Active              
p6jpyopzzy0zg4znegi63hzjq     worker5             Ready               Active              
yitkfnk99ecisrny9g3r9kfhk     worker3             Ready               Active              
yuwpmvtfmdxn8i7nllkyzkxkp *   manager1            Ready               Active              Leader
```

et que sur les 6 VM Docker, il y a 8 conteneurs,

```
docker@manager1:/$ docker service ps web
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE
ERROR               PORTS
jkm2hapcthht        web.1               nginx:latest        worker3             Running             Running 25
```

```
minutes ago
q55eqdhr1qf1        web.2               nginx:latest        worker4             Running             Running 25
minutes ago
imqdkw4ei6gs        web.3               nginx:latest        manager1            Running             Running 25
minutes ago
k4vjd0g7ijww        web.4               nginx:latest        worker1             Running             Running 25
minutes ago
b7xbmy1npgf9        web.5               nginx:latest        worker2             Running             Running 25
minutes ago
kg3bivcg0wln        web.6               nginx:latest        worker5             Running             Running 5
minutes ago
ik3u0jfgey64        web.7               nginx:latest        worker5             Running             Running 5
minutes ago
6bw5ptw7xao8        web.8               nginx:latest        worker1             Running             Running 5
minutes ago
```

dont deux se trouvent sur worker1 :

```
docker@manager1:/$ docker node ps worker1
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE
ERROR               PORTS
k4vjd0g7ijww        web.4               nginx:latest        worker1             Running             Running 26
minutes ago
6bw5ptw7xao8        web.8               nginx:latest        worker1             Running             Running 6
minutes ago
```

Mettez worker1 en mode d'indisponibilité en utilisant l'option **–availability drain** :

```
docker@manager1:/$ docker node update --availability drain worker1
worker1
```

Constatez que le service web a été déplacé sur deux autres noeuds, **manager1** et **worker4** :

```
docker@manager1:/$ docker service ps web
```

```
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE
ERROR               PORTS
jkm2hapcthht        web.1               nginx:latest        worker3             Running             Running 29
minutes ago
q55eqdhr1qf1        web.2               nginx:latest        worker4             Running             Running 29
minutes ago
imqdkw4ei6gs        web.3               nginx:latest        manager1            Running             Running 29
minutes ago
6cv6j4tz0nk5        web.4               nginx:latest        manager1            Running             Running 33
seconds ago
k4vjd0g7ijww         \_ web.4           nginx:latest        worker1             Shutdown            Shutdown 33
seconds ago
b7xbmy1npgf9        web.5               nginx:latest        worker2             Running             Running 29
minutes ago
kg3bivcg0wln        web.6               nginx:latest        worker5             Running             Running 9
minutes ago
ik3u0jfgey64        web.7               nginx:latest        worker5             Running             Running 9
minutes ago
wht3r8c9wga6        web.8               nginx:latest        worker4             Running             Running 33
seconds ago
6bw5ptw7xao8         \_ web.8           nginx:latest        worker1             Shutdown            Shutdown 33
seconds ago
```

3.10 - Supprimer un Service

Pour supprimer un service il convient d'utiliser la commande **docker service rm**

```
docker@manager1:/$ docker service rm web
web
```

```
docker@manager1:/$ docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
```

```
docker@manager1:/$ docker service inspect web
[]
Status: Error: no such service: web, Code: 1
```

Sortez de manager1 et démarrez le serveur VNC dans la machine virtuelle **debian9** en tant que **trainee** :

```
docker@manager1:/$ exit
root@debian9:~# exit
trainee@debian9:~$ vncserver

New 'X' desktop is debian9.i2tch.loc:1

Starting applications specified in /home/trainee/.vnc/xstartup
Log file is /home/trainee/.vnc/debian9.i2tch.loc:1.log
```

3.11 - Sauvegarder Docker Swarm

La configuration de Docker Swarm est contenue dans le répertoire **/var/lib/docker/swarm** de chaque Manager dans le Swarm. Ce processus necéssite qu'il y ait au moins **deux** Managers dans le Swarm. Le procédure de sauvegarde est :

- arrêt du service Docker sur le Manager à sauvegarder,
- sauvegarde du répertoire **/var/lib/docker/swarm**,
- redémarrage du service Docker sur le Manager concerné.

3.12 - Restaurer Docker Swarm

Le procédure de resturation est :

- arrêt du service Docker sur un nouveau Manager,
- suppression du contenu du répertoire **/var/lib/docker/swarm** dans le nouveau Manager,
- restauration du répertoire **/var/lib/docker/swarm** dans le nouveau Manager à partir de la sauvegarde,
- exécution de la commande **docker swarm init –force-new-cluster** sur le nouveau Manager,

* ajout des Managers et Workers à Swarm.
