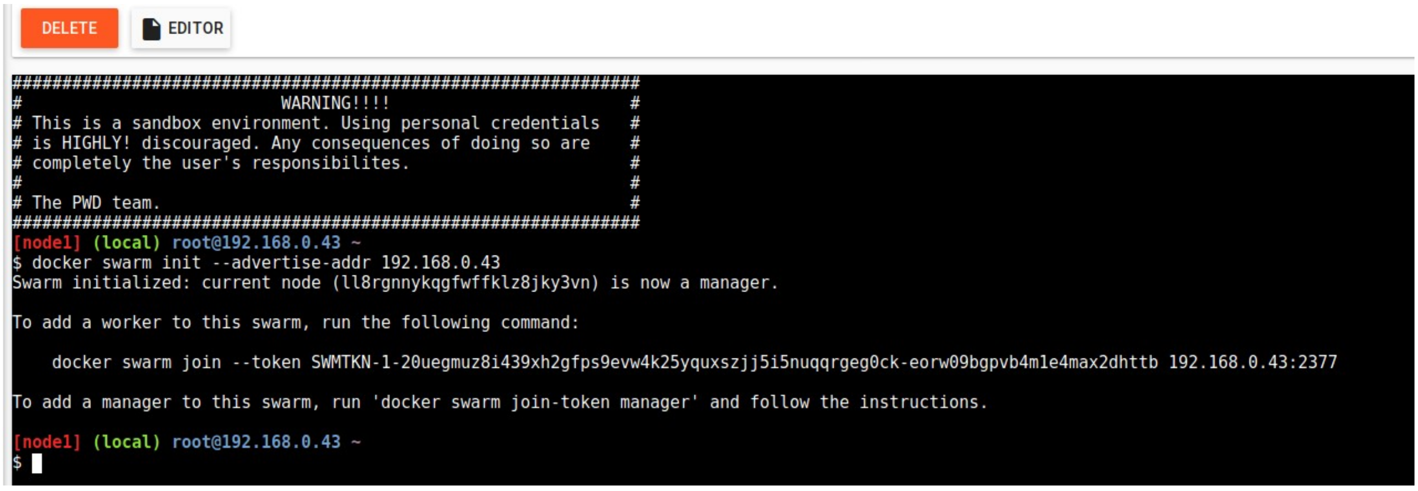
Cloud Orchestration with Docker Swarm

First Docker Swarm cluster

To initialize a swarm cluster we start to create a manager :

$ docker swarm init --advertise-addr <ip-adress>

The result of this command will generate a token that we can copy in others nodes:



```
###############################################################
#                          WARNING!!!!                        #
# This is a sandbox environment. Using personal credentials   #
# is HIGHLY! discouraged. Any consequences of doing so are    #
# completely the user's responsibilites.                      #
#                                                             #
# The PWD team.                                               #
###############################################################
[node1] (local) root@192.168.0.43 ~
$ docker swarm init --advertise-addr 192.168.0.43
Swarm initialized: current node (ll8rgnnykqgfwffklz8jky3vn) is now a manager.

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

    docker swarm join --token SWMTKN-1-20uegmuz8i439xh2gfps9evw4k25yquxszjj5i5nuqqrgeg0ck-eorw09bgpvb4m1e4max2dhttb 192.168.0.43:2377

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

[node1] (local) root@192.168.0.43 ~
$ 
```

So in PWD, we need to create others nodes (“add new instances”) and copy the token generated for the docker swarm join.

To visualize all nodes (Managers and workers) of the cluster (in manager node) :
$ docker node ls

Creating a service in a docker swarm cluster

1) Create a swarm cluster composed of 3 nodes : a manager and 2 workers.

2) On the manager, we will create a service to run an nginx container:

$ docker service create --name myNginxService -p 80:80 nginx you can verify the running of the service with the following command:

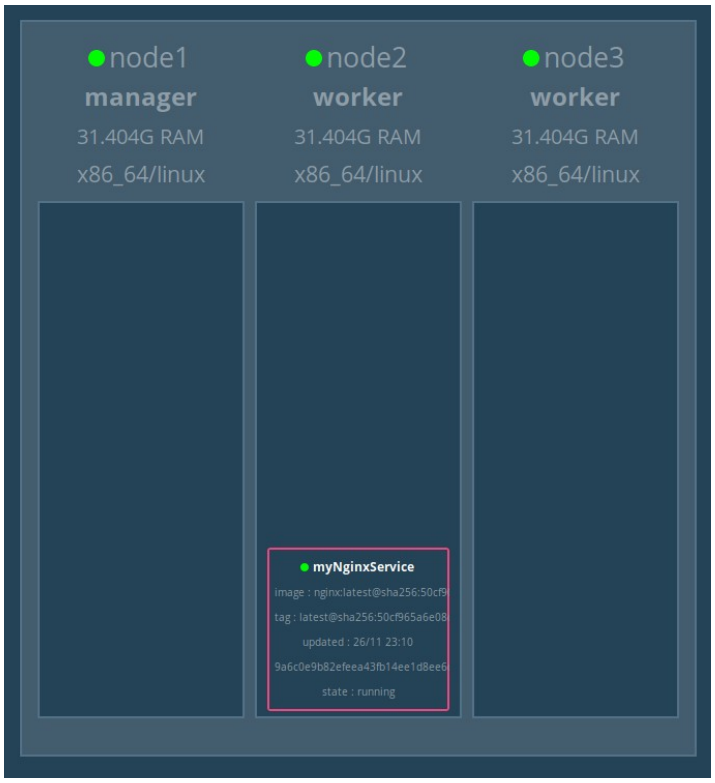
```
$ docker  service ls
```

we can click on port “80” of the manager instance and print the welcome page of nginx!

So, we can start to visualize graphically the nodes of the cluster, with this command on the node manager (it’s a dedicated docker’s container) :

```
$ docker run -it -d -p 8080:8080 -v /var/run/docker.sock:/var/run/docker.sock
dockersamples/visualizer
```

The result can be visualized clicking on port “8080” :



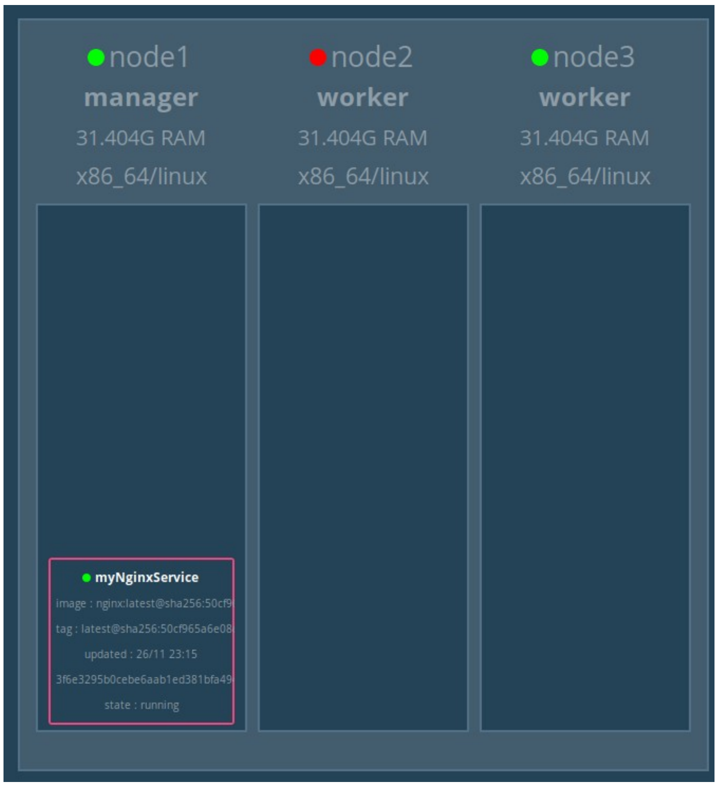
● node1
manager
31.404G RAM
x86_64/linux

● node2
worker
31.404G RAM
x86_64/linux

● node3
worker
31.404G RAM
x86_64/linux

● myNginxService
image : nginx:latest@sha256:50cf9
tag : latest@sha256:50cf965a6e08
updated : 26/11 23:10
9a6c0e9b82efeea43fb14ee1d8ee6
state : running

We can observe that the cluster is composed of 3 nodes, 1 manager and 2 workers, and the service is running in a worker node.
At this point the interest of a swarm cluster seems null compared to running a simple container.
But, what will happened if we delete the node number 2 ?
We observe that the service run on another node :



__**Deploying a stack on a docker swarm cluster**__

A stack is a composition of different services (more than one, like with docker compose).

In this example, we will make a service composed by a Flask host connected to a Redis database. The Flask host receive html requests (*post* and *get)*, and we will save the requests in a Redis database.

We want to run this docker-compose file:

```
------------------------------------------
version: '3'
services:
  app:
    build: .
    image: flask-redis:1.0
    environment:
      - FLASK_ENV=development
    ports:
      - 80:80
  redis:
    image: redis:4.0.11-alpine
------------------------------------------
```

All files needed are on DVO (docker-compose.yml, dockerfile, dependencies...)

Warning : we can’t build directly inside a docker-compose if we want to make a service.
The reason is that the normal usage is to separate creation of images, composition and deployment.

So you have to create the image with *flask-redis:1.0 name before (and suppress the build line).*

To copy the files to PWD, 2 possibilities :
 - take the ssh address and make a direct scp copy.
 - create and edit files online.

To create a file “myfile.txt” in command line :

```
$ touch myfile.txt
```

You can edit files with the “EDITOR” button.

Once the file shared to the cloud, we need to create the *flask-redis:1.0 and after deploy the docker-compose.yml*

we will create 3 nodes (1 manager and 2 workers).

Flask-redis:1.0 build

```
$ docker build -t flask-redis:1.0 .
```

Deployment of the service

```
$ docker stack deploy -c docker-compose.yml myService
```

To list running services:

```
$ docker service ls
```

To visualize the nodes of the clusters and containers distribution:

```
$ docker run -it -d -p 8080:8080 -v /var/run/docker.sock:/var/run/docker.sock
dockersamples/visualizer
```

To understand what this service realize, run the post-get.sh script file.
We need the execution privileges to run the script:

```
$ chmod u+x post-get.sh
```

To run the file:

```
$ ./post-get.sh
```

re-run several times this file and read carefully all files and explain exactly what this services does.