Centro de Procesamiento de Datos

Práctica 6. Docker Swarm: Combinando múltiples máquinas para la ejecución de contenedores Docker.

Objetivo:
Crear un entorno basado en tres máquinas virtuales con Vagrant (+ VirtualBox) y evaluar diversas configuraciones de ejecución de contenedores Docker.

Presentar un documento pdf en SWAD→Actividades→ Práctica 6 con la siguiente información:

-(obligatorio): Realizar diversas capturas donde se muestren:
-La creación de las máquinas virtuales.
-El inicio del manager de docker swarm. Ej:

```
To add a worker to this swarm, run the following command:
docker swarm join --token SWMTKN-1-
263n2zpxp5f5odhnqdcx67ejh199ig1lzlq62w88nsaue9smwk-eerzcjju0ukrrp22q3vnb7vfw
192.168.99.100:2377
```

- Capturas de las redes internas.
- Ejecución del servicio web
- Cuando los 3 nodos están activos
- Cuando se cambia de escala a 2
- Cuando apagamos un nodo activo y sólo ejecuta un nodo,
- y la activación automática del segundo nodo.

- (opcional): Capturas de diversas ejecuciones en la plataforma Katakoda.

Desarrollo:

En esta práctica estudiamos cómo automatizar la creación de máquinas virtuales con Vagrant y VirtualBox, creando un escenario con 3 máquinas virtuales.

Sobre estas máquinas instalamos dos servidores GlusterFS trabajando en modo replicado y accediendo desde un cliente.

I) Docker-machine

Docker Machine permite crear máquinas virtuales tipo virtualbox o equivalentes que actuarán como nodos de nuestra red.
Para instalar docker-machine seguir los siguientes pasos dependiendo del entorno de trabajo:
https://docs.docker.com/machine/install-machine/

En el caso de Linux, docker-machine utiliza VirtualBox, por lo que previamente debe estar instalador. En caso de que se esté ejecutando una máquina virtual no podemos anidar la virtualización (ejecutar VirtualBox dentro de una máquina VirtualBox), pero sí podemos crear contenedores LXD como los que vimos en la primera práctica.

La práctica la seguimos desarrollando con LXD si no es posible instalar docker-machine.
Instalación de docker en contenedores LXD:

```
docker launch ubuntu: u1 -c security.nesting=true
```

Una vez dentro del contenedor, instalamos docker con una versión reciente:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) edge"
apt update
apt install -y docker-ce
```

Si podemos hacerlo con docker-machine:
Creamos una máquina con:

```
docker-machine create m1
```

Podemos ver la lista de máquinas:

```
docker-machine ls
```

Entramos por ssh a la máquina:

```
Docker-machine ssh m1
```

Para conocer la IP de la máquina

```
Docker-machine ip m1
```

II) Evaluando Docker Swarm

Docker Swarm permite distribuir contenedores entre distintas máquinas de forma que pueda distribuirse la ejecución.

Para los siguientes apartados necesitamos 3 máquinas virtuales.

Dentro de la máquina m1 que hemos creado ejecutamos:

```
docker swarm init --advertise-addr 192.168.99.100
```

Esa IP es la dirección de la subred interna visible entre los nodos.
Obtenemos:

```
docker@m1:~$ docker swarm init --advertise-addr 192.168.99.100
Swarm initialized: current node (nu1lrod1xq93p4ag05i7cdzf3) is now a manager.

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

docker swarm join --token SWMTKN-1-263n2zpxp5f5odhnqdcx67ejh199ig1lzlq62w88nsaue9smwk-eerzcjju0ukrrp22q3vnb7vfw 192.168.99.100:2377

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

Comprobamos los nodos con:

```
docker node ls
```

Creamos un segundo nodo *m2* con docker-machine (o vagrant + virtualbox)

```
docker-machine create m2
```

Y entramos por ssh

```
docker-machine ssh m2
```

Y ejecutamos la orden que nos apareció para añadirlo al swarm:

```
docker swarm join --token SWMTKN-1-263n2zpxp5f5odhnqdcx67ejh199ig1lzlq62w88nsaue9smwk-eerzcjju0ukrrp22q3vnb7vfw 192.168.99.100:2377
```

En el nodo *m1* podemos comprobar que ya aparece el segundo nodo

```
docker node ls
```

Añadimos el tercer nodo *m3* .

III) Creamos un servicio

Vamos a lanzar un servicio web con 3 replicas que se distribuyen entre los nodos.

```
docker service create --name web  --replicas 3 --mount type=bind,src=/etc/hostname,dst=/usr/share/nginx/html/index.html,readonly --publish published=8080,target=80 nginx
```

De esta forma cuando accedemos a cualquiera de los tres nodos.

Con la red ingress que utiliza por defecto se define una red que balancea el tráfico y reencamina automáticamente entre los nodos.
Para más información sobre los diversos modelos de red en Docker:
https://docs.docker.com/v17.09/engine/swarm/networking/

Si estamos en nuestro host principal o bien desde cualquier nodo podemos ejecutar:

```
curl http://192.168.99.102:8080
```

Cada vez que lo ejecutamos podemos ver que responde un nodo distinto (m1, m2 o m3).

Podemos comprobar que el servicio ha lanzado 3 contenedores, uno en cada nodo.

```
docker service ps web
```

Reducimos el número de nodos:

```
docker service scale web=2
```

El sistema sigue funcionando y comprobamos el numero de contenedores

Forzamos una parada de uno de los nodos que sigue activo, ej.m3

```
docker-machine stop m3
```

Como sólo está funcionando un contenedor, a los pocos segundos se activará automáticamente el segundo contenedor. (Lo comprobamos de forma periódica).
Si reactivamos el nodo m3, podremos ver también que el nodo también aparece y si reescalamos a 3 contenedores activos aparece automáticamente en el nuevo nodo.

IV) Monitorizar Docker Swarm

En el nodo m1 desplegamos Swarmprom

```
$ git clone https://github.com/stefanprodan/swarmprom.git
$ cd swarmprom

ADMIN_USER=admin \
ADMIN_PASSWORD=admin \
SLACK_URL=https://hooks.slack.com/services/TOKEN \
SLACK_CHANNEL=devops-alerts \
SLACK_USER=alertmanager \
docker stack deploy -c docker-compose.yml mon
```

V) Control de ejecución

--cpu-limit=0.5
--cpu-quota=x000 (x%)
--cpuset="",,..."

Creamos un contenedor que nos permita crear consumir 1 CPU
Dockerfile

```
FROM ubuntu
RUN apt-get update
RUN apt-get install stress
CMD /usr/bin/stress --cpu 1
```

Comprobamos al lanzar la ejecución que podemos reducir el consumo de la CPU.

VI) Katakoda https://www.katacoda.com/

Katakoda es una plataforma de aprendizaje de diversos sistemas que requieren acceso remoto, generalmente en modo o terminal o a través.

Katakoda nos ofrece múltiples escenarios para ampliar nuestro conocimiento sobre contenedores y otras tecnologías.

En particular vamos a evaluar alguna de las actividades con Docker.