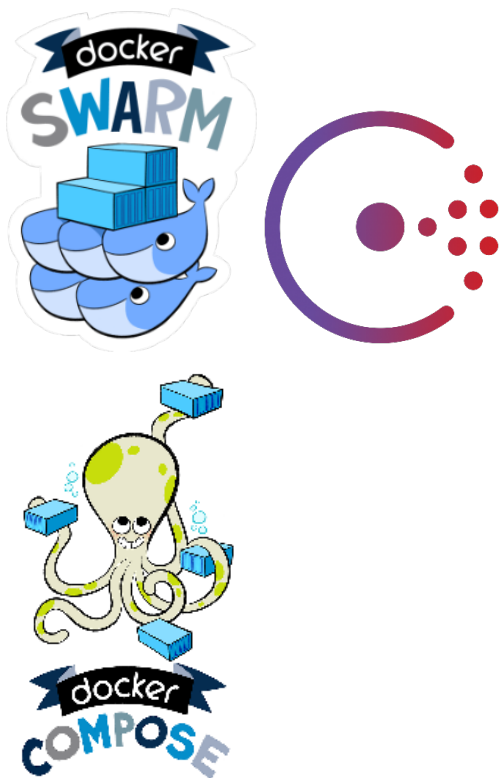


DOCKER-3 OpenCloud: Cap a models d'integració contínua basats en Docker



Continguts

- *Balanceig transparent amb Docker Swarm, Compose i Consul*
 - *Requeriments previs*
 - *Consul*
 - *Docker Swarm*
 - *Load Balancer*
 - *Docker Compose*
 - *Exemple d'ús*

Balanceig transparent amb Docker Swarm, Compose i Consul

Crearem un cluster de docker-engine i instanciem allà els nostres contenidors docker. A més configurarem un servei balancejat d'alta disponibilitat escalable.

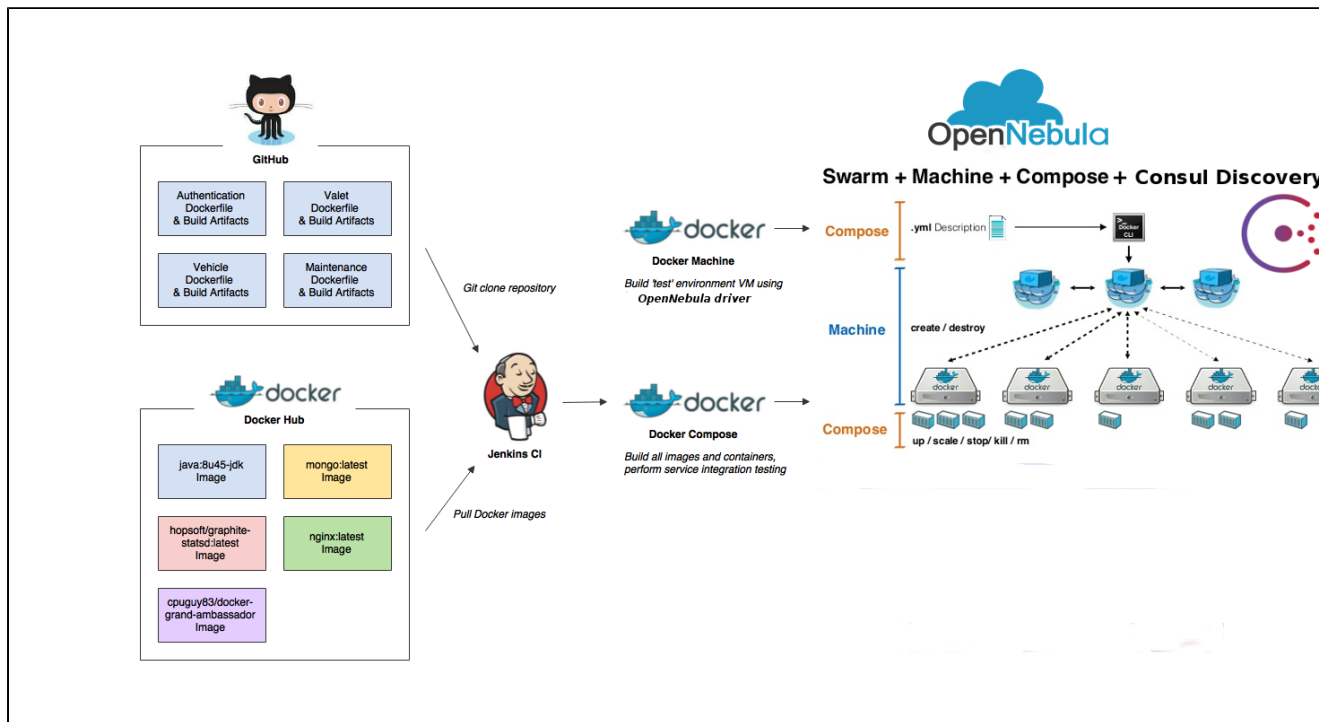
Docker Swarm: Ens permet crear un cluster docker. El node master rebrà i distribuirà els contenidors als nodes secundaris on s'executaran.

Consul: Ens proporcionarà el servei de discover dels nous nodes i serveis que afegim al clúster swarm.

Registrar: Aquest servei monitoritzarà la informació com ara IP i PORT de cada servei activat als hosts i ho guardarà al consul.

Docker Compose: Ens permet configurar i crear grups de contenidors. Podem escalar el servei ofert per un contenir augmentar o disminuint el nombre de contenidors.

En la següent imatge es mostren un seguit **eines d'DevOps** cobreix tot el cicle de vida de les aplicacions, en el tutorial ens centrarem en la part del desplegament fent servir la plataforma **OpenNebula** juntament amb **Docker Swarm**, **Docker Compose** i el servei de **Consul**.



Requeriments previs

- **Docker Engine:** Ens permet crear els contenidors, gestionar-los i instanciar-los
- **Docker Machine:** Ens permet crear i gestionar les màquines virtuals
- **Driver Docker Machine Opennebula:** Ens permet utilitzar el docker-machine al cloud de l'OpenNebula
- **Docker Compose**

Instal·lar el requeriments:

- Per a poder implementar el **docker swarm cluster** amb l'OpenNebula hem d'instalar el **Docker Engine**, **Docker Machine** i el **Driver Docker Machine Opennebula**. Aquí està el manual de com fer-ho: *DOCKER-1 Docker Machine OpenNebula Driver*
- Per instal·lar **Docker Compose**:

```
sudo -i
curl -L https://github.com/docker/compose/releases/download/1.6.2/docker-compose-`uname -s`-
`uname -m` > /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
exit
```

Consul

Disposarem d'una màquina virtual que ens proporcionarà el servei de discover amb consul

Creem una màquina virtual a l'OpenNebula anomenada consul i després executem un contenidor docker de consul dins d'ella.

```
docker-machine create -d opennebula --opennebula-network-id [network_id] --opennebula-image-id
[boot2docker_image_id] --opennebula-b2d-size [volatile_disk_size] consul
docker $(docker-machine config consul) run -d -p "8500:8500" -h "consul" progrim/consul -server -
bootstrap
```

Docker Swarm

Diposarem dos tipus de màquines virtuals amb swarm:

- **Swarm-master:** Aquesta màquina amb swarm serà l'encarregada de distribuir les instàncies dels contenidors als diferents nodes que tinguem al clúster.
- **Swarm-node:** Aquesta o aquestes màquines seran les que executaran els contenidors docker.

Swarm-master

Creem la màquina virtual del swarm-master:

```
export CONSUL_IP=$(docker-machine ip consul)
docker-machine create -d opennebula --opennebula-network-id [network_id] --opennebula-image-id
[boot2docker_image_id] --opennebula-b2d-size [volatile_disk_size] --swarm --swarm-master --swarm-
discovery="consul://$CONSUL_IP:8500" --engine-opt cluster-store=consul://$CONSUL_IP:8500 --engine-opt
cluster-advertise="eth0:2376" swarm-master
```

Swarm-node

Creem la màquina virtual del swarm-node:

```
docker-machine create -d opennebula --opennebula-network-id [network_id] --opennebula-image-id
[boot2docker_image_id] --opennebula-b2d-size [volatile_disk_size] --swarm --swarm-discovery="
consul://$CONSUL_IP:8500" --engine-opt cluster-store=consul://$CONSUL_IP:8500 --engine-opt cluster-
advertise="eth0:2376" swarm-node-1
```

Podem crear els nodes que vulguem. Només hem de canviar el nom de la màquina virtual

Desplegar contenidors Registrator

Un cop hem creat el master i els diferents nodes, instanciem els contenidors amb registrator a tots ells. Utilitzarem la imatge de *gliderlabs/registrator*

Necessitem tenir un servei Registrator corrent en cada host per monitoritzar tots els serveis que corre cada host.

```
export MASTER_IP=$(docker-machine ip swarm-master)
export NODE_IP=$(docker-machine ip swarm-node-1)

eval $(docker-machine env --swarm swarm-master)
docker run -d --name=registrator -h ${MASTER_IP} --volume=/var/run/docker.sock:/tmp/docker.sock
gliderlabs/registrator:latest consul://${CONSUL_IP}:8500

eval $(docker-machine env swarm-node-1)
docker run -d --name=registrator -h ${NODE_IP} --volume=/var/run/docker.sock:/tmp/docker.sock gliderlabs
/registrator:latest consul://${CONSUL_IP}:8500
```

Load Balancer

Ara hem d'implementar un balancejador de càrrega que pugui distribuir les peticions per les diferents instàncies del servei. Com augmentarem i disminuïrem les instàncies del servei, necessitem que el balancejador de càrrega s'actualitzi automàticament.

Utilitzarem nginx per al load balancer i consul-template per la configuració del nginx.

default.ctmpl

Primer de tot, necessitem crear un template per a la configuració del nginx. Consul-template automàticament omplirà aquest fitxer amb la informació referent al servei instanciat i crearà la configuració pel nginx.

El fitxer `default.ctmpl` ha de ser així:

```
{{ $app := env "APP_NAME" }}

upstream {{ printf $app }} {
    least_conn;
    {{ range service $app }}
        server {{ .Address }}:{{ .Port }} max_fails=3 fail_timeout=60 weight=1; {{ end }}
}

server {
    listen 80 default;

    location / {
        proxy_pass http://{{ printf $app }};
    }
}
```

Crearem la variable **app** amb el valor de la variable d'entorn APP_NAME (definida més endavant al fitxer docker-compose.yml). També crearem un upstream amb el nom de la variable **app**. La línia *least_conn* fa que nginx encamini el traffic cap a l'instància amb menys connexions.

Per cada instància del servei corrent crearem una línia amb l'adreça del node on s'executa ({{.Address}}) i el port per on està escoltant ({{.Port}}).

Finalment està la zona de configuració del servidor. Aquí definim el port d'escolta del load balancer i creem una reverse proxy a l'upstream que hem creat.

start.sh

Necessitem un script que actui com a entry point per a aquesta imatge docker.

El fitxer `start.sh` ha de ser així:

```
#!/bin/bash
service nginx start
consul-template -consul=$CONSUL_URL -template="/templates/default.ctmpl:/etc/nginx/conf.d/default.conf:
service nginx reload"
```

Aquest script engega primer el servei nginx. Després engega consul-template passant-li dos paràmetres:

- -consul: Li passem la variable d'entorn \$CONSUL_URL definida al docker-compose.yml
- -template: Li passem 3 paràmetres: el path on està el template que hem creat anteriorment (dins del contenidor), el path on es guardarà el fitxer de configuració generat i la comanda que s'executarà un cop es generi una nova configuració.

El consul-template crearà un nou fitxer de configuració cada cop que un servei s'engegui o s'aturi. La informació sobre aquests serveis es recollida per el servei **registrator** de cada node swarm i es emmagatzemada en el consul.



El fitxers `start.sh` i `default.ctmpl` els guardarem en un directori anomenat "files". En el directori pare hi guardarem els fitxers **Dockerfile** i **docker-compose.yml**

Dockerfile

```
FROM nginx:latest

RUN apt-get update \
    && apt-get install -y unzip

ADD files/start.sh /bin/start.sh
RUN chmod +x /bin/start.sh
ADD files/default.ctmpl /templates/default.ctmpl

ADD https://releases.hashicorp.com/consul-template/0.12.2/consul-template_0.12.2_linux_amd64.zip /usr
/bin/
RUN unzip /usr/bin/consul-template_0.12.2_linux_amd64.zip -d /usr/local/bin

EXPOSE 80
ENTRYPOINT ["/bin/start.sh"]
```

Aquest Dockerfile utilitza nginx com a imatge base i instal·la consul-template damunt. Després copia l'script start.sh i el template default.ctmpl (que abans hem creat) dins del contenidor. Finalment exposa el port 80 i defineix el script start.sh com a entry point de la imatge.

Docker Compose

Docker Compose ens permet escriure la configuració que volem que tinguin els contenidors a desplegar. Utilitzarem Docker Compose File version 2, que ens permet definir la configuració referent a la xarxa, volums, ports, variables d'entorn.

Per saber més sobre la versió 2 del fitxer de docker-compose, vés [aquí](#).

docker-compose.yml

Exemple de docker-compose.yml

```

version: '2'
services:
  lb:
    build: .
    container_name: lb
    ports:
      - "80:80"
    environment:
      - APP_NAME=[Nom_del_servei]
      - CONSUL_URL=${CONSUL_IP}:8500
    depends_on:
      - web
    networks:
      - front-tier
  web:
    image: [imatge_docker]
    ports:
      - "[Port_del_servei]"
    environment:
      - SERVICE_NAME=[Nom_del_servei]
    networks:
      - front-tier
networks:
  front-tier:
    driver: overlay

```

Gestionar Docker Compose

Per arrencar tots el serveis executem el següent.

Per a veure els detalls dels serveis corrent podem utilitzar la comanda *docker-compose ps*

```

eval $(docker-machine env -swarm swarm-master)
docker-compose up -d
docker-compose ps

```

Per parar i eliminar els serveis que estan corrent actualment:

```

docker-compose stop; docker-compose rm -f

```

Ara mateix només tenim una instància del servei. Si volem augmentar o disminuir

```

docker-compose scale [nom_servei]=[nombre d'instàncies]

```

Per a més informació : <https://botleg.com/stories/load-balancing-with-docker-swarm/>

Exemple d'ús

Volem crear un cluster de docker i crear un servei web balancejat amb nginx i escalable amb docker Compose.

Actualment disposeu de dues imatges, ja precreades amb el Docker Engine instal·lat que podeu fer servir:

- *boot2docker*
- Docker-Machine-Ubuntu-14.04

- **Creem les màquines virtuals de consul, swarm-master i un swarm-node:**

```
docker-machine create -d opennebula --opennebula-network-id $NETWORK_ID --opennebula-image-name boot2docker --opennebula-b2d-size 10240 consul
docker $(docker-machine config consul) run -d -p "8500:8500" -h "consul" progrium/consul -server -bootstrap
export CONSUL_IP=$(docker-machine ip consul)
docker-machine create -d opennebula --opennebula-network-id $NETWORK_ID --opennebula-image-name boot2docker --opennebula-b2d-size 10240 --swarm --swarm-master --swarm-discovery="consul://$CONSUL_IP:8500" --engine-opt cluster-store=consul://$CONSUL_IP:8500 --engine-opt cluster-advertise="eth0:2376" swarm-master
docker-machine create -d opennebula --opennebula-network-id $NETWORK_ID --opennebula-image-name boot2docker --opennebula-b2d-size 10240 --swarm --swarm-discovery="consul://$CONSUL_IP:8500" --engine-opt cluster-store=consul://$CONSUL_IP:8500 --engine-opt cluster-advertise="eth0:2376" swarm-node-1
```

--opennebula-b2d-size 10240: Creem discos volàtils de 10GB

❗ On \$NETWORK_ID, serà l'ID de la xarxa on crearem el clúster.

Virtual Machines

☐	ID	Owner	Group	Name	Status	Host	IPs	
☐	2265		Docker	swarm-node-01	RUNNING	cluster06		
☐	2264		Docker	swarm-master	RUNNING	cluster04		
☐	2261		Docker	consul	RUNNING	cluster04		

Showing 1 to 3 of 3 entries

Previous 1 Next 10

3 TOTAL 3 ACTIVE 0 OFF 0 PENDING 0 FAILED

- **Despleguem Registrator a tots els nodes**

```
export MASTER_IP=$(docker-machine ip swarm-master)
export NODE_IP=$(docker-machine ip swarm-node-1)

eval $(docker-machine env --swarm swarm-master)
docker run -d --name=registrator -h ${MASTER_IP} --volume=/var/run/docker.sock:/tmp/docker.sock
gliderlabs/registrator:latest consul://${CONSUL_IP}:8500

eval $(docker-machine env swarm-node-1)
docker run -d --name=registrator -h ${NODE_IP} --volume=/var/run/docker.sock:/tmp/docker.sock
gliderlabs/registrator:latest consul://${CONSUL_IP}:8500
```

- **Creem els fitxers de configuració necessaris**

files/default.ctmpl

```
{{ $app := env "APP_NAME" }}

upstream {{ printf $app }} {
    least_conn;
    {{ range service $app }}
    server {{ .Address }}:{{ .Port }} max_fails=3 fail_timeout=60 weight=1;{{ end }}
}

server {
    listen 80 default;

    location / {
        proxy_pass http://{{ printf $app }};
    }
}
```

files/start.sh

```
#!/bin/bash
service nginx start
consul-template -consul=$CONSUL_URL -template="/templates/default.ctmpl:/etc/nginx/conf.d/default.conf:
service nginx reload"
```

Dockerfile

```

FROM nginx:latest

RUN apt-get update \
    && apt-get install -y unzip

ADD files/start.sh /bin/start.sh
RUN chmod +x /bin/start.sh
ADD files/default.ctmpl /templates/default.ctmpl

ADD https://releases.hashicorp.com/consul-template/0.12.2/consul-template_0.12.2_linux_amd64.zip /usr
/bin/
RUN unzip /usr/bin/consul-template_0.12.2_linux_amd64.zip -d /usr/local/bin

EXPOSE 80
ENTRYPOINT ["/bin/start.sh"]

```

docker-compose.yml

```

version: '2'
services:
  lb:
    build: .
    container_name: lb
    ports:
      - "80:80"
    environment:
      - APP_NAME=web_nginx
      - CONSUL_URL=${CONSUL_IP}:8500
    depends_on:
      - web
  web:
    image: nginx
    ports:
      - "80"
    environment:
      - SERVICE_NAME=web_nginx
networks:
  default:
    driver: overlay

```

- **Activem el servei**

Des del directori on tenim el docker-compose.yml, executem:

```

eval $(docker-machine env -swarm swarm-master)
docker-compose up -d
docker-compose ps

```

```

Creating network "swarmcompose_default" with driver "overlay"
Pulling web (nginx:latest)...
swarm-node-1: Pulling nginx:latest... : downloaded
swarm-master: Pulling nginx:latest... : downloaded
Building lb
Step 1 : FROM nginx:latest
--> eb4a127a1188
Step 2 : RUN apt-get update && apt-get install -y unzip
--> Running in 3d6a67072f9b
Get:1 http://security.debian.org jessie/updates InRelease [63.1 kB]
Ign http://nginx.org jessie InRelease
Ign http://httpredir.debian.org jessie InRelease
Get:2 http://security.debian.org jessie/updates/main amd64 Packages [291 kB]
Get:3 http://nginx.org jessie Release.gpg [287 B]
Get:4 http://httpredir.debian.org jessie-updates InRelease [142 kB]
Get:5 http://nginx.org jessie Release [2325 B]
Get:6 http://nginx.org jessie/nginx amd64 Packages [7377 B]
Get:7 http://httpredir.debian.org jessie Release.gpg [2373 B]
Get:8 http://httpredir.debian.org jessie Release [148 kB]
Get:9 http://httpredir.debian.org jessie-updates/main amd64 Packages [7407 B]
Get:10 http://httpredir.debian.org jessie/main amd64 Packages [9034 kB]
Fetched 9698 kB in 20s (480 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
Suggested packages:
  zip
The following NEW packages will be installed:
  unzip
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 162 kB of archives.
After this operation, 347 kB of additional disk space will be used.
Get:1 http://httpredir.debian.org/debian/ jessie/main unzip amd64 6.0-16+deb8u2 [162 kB]
debconf: delaying package configuration, since apt-utils is not installed
Fetched 162 kB in 1s (97.2 kB/s)
Selecting previously unselected package unzip.
(Reading database ... 9733 files and directories currently installed.)
Preparing to unpack ../unzip_6.0-16+deb8u2_amd64.deb ...
Unpacking unzip (6.0-16+deb8u2) ...

```

```

root@swarm-master: ~/Escriptori/swarm+compose$ docker-compose ps

```

Name	Command	State	Ports
lb	/bin/start.sh	Up	443/tcp, :80->80/tcp
swarmcompose_web_1	nginx -g daemon off;	Up	443/tcp, :32768->80/tcp

• Augmentar o disminuir las instancias del servei

Per augmentar o disminuir les instàncies del servei utilitzem la següent comanda

```
docker-compose scale web=4
```

```
colom@nas001: ~/Escriptori/swarm+compose$ docker-compose scale web=4
Creating and starting swarmcompose_web_2 ... done
Creating and starting swarmcompose_web_3 ... done
Creating and starting swarmcompose_web_4 ... done
```

Amb aquest comanda augmentarem en 4 les instàncies desplegades del servei web.

El nom que indiquem ha de ser el nom que hem indicat en el fitxer docker-compose.yml. Nosaltres el fitxer docker-compose.yml hem indicat que el nom del servei era **web**.

Executem **docker-compose ps** per saber com estan les instàncies. Hem esborrat les IP. Aquí es veurien com les instàncies estan repartides pel diferents nodes del cluster swarm.

```
colom@nas001: ~/Escriptori/swarm+compose$ docker-compose ps
```

Name	Command	State	Ports
lb	/bin/start.sh	Up	443/tcp, :80->80/tcp
swarmcompose_web_1	nginx -g daemon off;	Up	443/tcp, :32768->80/tcp
swarmcompose_web_2	nginx -g daemon off;	Up	443/tcp, :32769->80/tcp
swarmcompose_web_3	nginx -g daemon off;	Up	443/tcp, :32768->80/tcp
swarmcompose_web_4	nginx -g daemon off;	Up	443/tcp, :32769->80/tcp

- **Accedir al servei**

Hem d'accedir amb la IP del **load balancer (lb)**

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

- **Aturar i/o esborrar el serveis**

```
docker-compose stop; docker-compose rm -f
```

