Clúster Autoescalable amb Swarm

Moya Ahufinger, Oriol

Guim Bernat, Francesc

Universitat Oberta de Catalunya

Índex

Introducció 3
Motivació..... 4
Descripció del projecte..... 6
Objectius i Resultats..... 6
Anàlisi de riscos..... 6
Abast de la proposta 7
Organització del Projecte 9
Pla de treball 10
Relació d'activitats 10
Fites principals..... 10
Calendari de treball..... 10
Valoració Econòmica..... 13
Arquitectura 15
Servidor de Control 16
Start Container 17
Start Server..... 20
Destroy Server..... 21
Check CPU 23
NGINX Proxy..... 26
Web UOC Container..... 33
Swarm Master 38
Swarm Node..... 39
Proves de Rendiment..... 40
Millores..... 49
Rendiment..... 51
Valoració..... 54
Annex A 56
Referències..... 59

Introducció

Aquest projecte pretén aprofundir en les noves tecnologies relacionades amb la utilització de contenidors amb Docker. Un contenidor permet encapsular tot el software necessari per poder executar una aplicació; conté el sistema operatiu, les aplicacions de tercers i els binaris del nostre programa. D'aquesta manera és possible crear el contenidor i executar-lo en l'entorn de desenvolupament, després en el de test i finalment a producció garantint que sempre estem executant el mateix codi i amb les mateixes versions en totes les plataformes.

Aquesta nova tecnologia pretén anar un pas més enllà de la virtualització clàssica on el que es virtualitza és tot un servidor. En aquest cas fem petites virtualitzacions que contenen tot el necessari per executar el codi i s'empaqueten en contenidors que podem executar en qualsevol servidor.

D'aquesta manera és molt fàcil desplegar una aplicació ja que només hem d'instal·lar el servidor base amb Docker i executar el contenidor. Com que aquest conté tots els components necessaris es garanteix el seu funcionament i no cal fer la instal·lació de cada un dels components de software.

Aquest sistema dóna una gran flexibilitat i redueix molt el temps de posada a producció de qualsevol programari. També garanteix que les proves fetes en els altres entorns, com el de desenvolupament, test i preproducció, seran vàlides ja que el que acabarem executant serà exactament el mateix. Així evitem els típics problemes entre les diferents configuracions dels diferents entorns. Per aquest motiu creiem que Docker és la millor opció per desenvolupar aquest projecte.

En aquest document descriurem la motivació i abast d'aquest projecte, també la definició de tasques i el cost que d'aquestes es deriva.

Motivació

Cada dia les empreses miren de poder reduir els costos que es deriven del manteniment de tots els seus servidors i dels CPD's. Per això, les empreses estan externalitzant els seus centres de dades i estan apostant pel model al núvol.

És en aquest context on mirar d'optimitzar al màxim els recursos que s'utilitzen és primordial per les empreses. En aquest nou model no es paga per un servidor físic sinó per l'ús que fas dels recursos proporcionats. Aquests són bàsicament disc, memòria i cpu.

Per tant, si som capaços d'adaptar l'ús dels recursos a les nostres necessitats es podrà aconseguir un guany econòmic ja que només pagarem per les necessitats de cada moment.

Per poder realitzar aquest projecte ens hem plantejat dues opcions. Es pot realitzar amb virtualització clàssica o amb contenidors.

La principal diferència entre els dos sistemes és que la virtualització crea servidors amb el seu propi S.O. de manera totalment aïllada i els contenidors comparteixen el mateix kernel i S.O. entre ells. Les parts compartides són només de lectura i després tenim les pròpies de cada contenidor que només són accessibles per ell mateix.

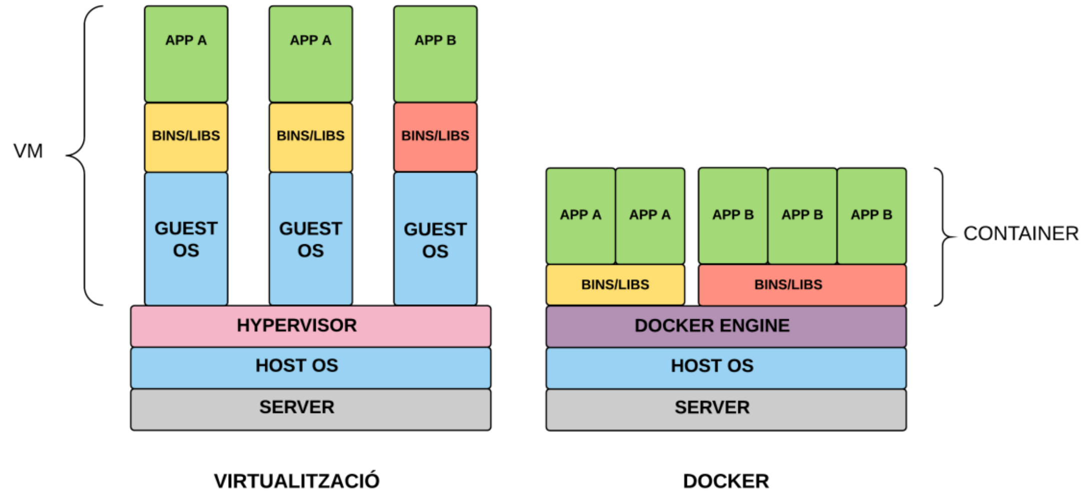

Figura 1 - Comparativa Virtualització vs Containers

Això té avantatges en l'optimització de recursos. Si tenim una MV que ocupa 15 Gb i la volem replicar en el mateix servidor necessitem 150Gb per allotjar deu màquines ja que cada una d'elles és totalment independent de l'altre. En canvi, en el cas dels contenidors, com que tenim parts comunes que es reutilitzen tindrem un estalvi considerable d'espai.

Un altre punt a tenir en compte és que les MV poden executar qualsevol S.O. i per tant qualsevol tipus d'aplicació. En els contenidors com que estan basats en LXC (Linux Containers) comparteixen el mateix kernel que el host i per tant només poden executar aplicacions Linux compatibles amb aquest Host. Aquest, actualment, és el principal inconvenient ja que limita el tipus d'aplicacions que podem executar.

Les principals diferències són les següents:

Virtualització	**Docker**
Màquina Virtual	Contenidor
Template	Image
Arrenca en minuts	Arrenca en segons
Pot executar qualsevol S.O.	Limitat a Linux
Les MV depenen del Hypervisor	Els contenidors són portables
Les MV ocupen varis Gb	Els contenidors són molt més lleugers

Taula 1 – Comparativa Virtualització vs Dockers

Un cop analitzades les diferències entre la virtualització i l'ús de contenidors veiem que són tecnologies complementaries. Podem executar contenidors en màquines virtuals i és aquest fet el que ens permet realitzar aquest projecte de la manera més simple. A l'unir aquestes dos tecnologies podem crear servidors virtuals de forma fàcil i ràpida i després desplegar les aplicacions que necessitem. Per tant, podem aconseguir un sistema que de forma dinàmica s'adapti a les necessitats de les empreses. D'aquesta manera podem reduir costos sense limitar les possibilitats de créixer ni el rendiment dels serveis que s'ofereixen.

Aquest projecte defineix una arquitectura capaç de créixer o disminuir depenent de la càrrega que té el sistema. Garantint així un bon nivell de servei amb el mínim cost possible.

Descripció del projecte

Amb aquest projecte pretenem crear un sistema dinàmic que sigui capaç de dimensionar-se segons la càrrega rebuda. D'aquesta manera es pretén assolir els conceptes comentats en els apartats anteriors amb els beneficis que això comporta.

Per fer-ho farem una planificació i un disseny de les tasques i components que volem implantar tenint en compte les tecnologies actuals.

Finalment portarem a la pràctica aquest disseny per comprovar de forma empírica que tot el que es volia aconseguir realment ens aporta els beneficis descrits.

Objectius i Resultats

Enumerarem a continuació els principals objectius d'aquest projecte:

- Analitzar les noves tecnologies relacionades amb els contenidors de software. Docker, Docker Swarm i Docker Machine.
- Aprendre com s'instal·len i es configuren.
- Verificar que realment fan allò que diuen tot i estar en fase Beta.
- Demostrar que poden servir per ajudar a abaratir costos.

Anàlisi de riscos

Com a tot projecte on els recursos temporals i econòmics són limitats, hem de minimitzar el possible impacte que pugui produir una situació de risc no desitjada. Per tant es prendran les següents mesures preventives i de contingència:

- Realitzar un seguiment setmanal de l'estat del projecte segons la seva planificació inicial, posant en comú els aspectes tractats i prioritzant les tasques amb acabament més imminent.
- Assegurar la disponibilitat del servei de hosting online on es realitzarà la instal·lació i les proves de funcionament dels sistema.
- Triar correctament els perfils de l'equip de treball i evitar rotacions d'aquests membres durant la vida del projecte.
- Emmarcar i respectar acuradament l'abast del projecte.

Abast de la proposta

Les tecnologies que pretenem analitzar són:

- CoreOS[1]: Un nou sistema operatiu creat per poder executar contenidors de la manera més òptima.
- Docker[2]: Software que permet crear contenidors amb les aplicacions que es vulgui.
- Docker Swarm[3]: Software que permet gestionar diferents contenidors que s'executen en màquines diferents com si estiguessin en la mateixa.
- Docker Machine[4]: Software que permet desplegar en entorns virtuals noves màquines ja configurades a partir d'un script.
- Nginx[5]: Proxy que ens permet redireccionar el tràfic d'entrada cap a altres servidors. Té altres funcionalitats però en aquest projecte només avaluarem aquesta.

El projecte estarà definit per una granja de servidors. Aquests servidors estaran dockeritzats amb la tecnologia Docker. Per davant tindran un servidor amb un proxy Nginx que distribuirà la càrrega. Les màquines on s'executaran aquests contenidors tindran el S.O. CoreOS. Per poder gestionar els diferents contenidors entre les màquines utilitzarem Docker Swarm. I per poder desplegar més servidor de forma automàtica farem servir Docker Machine.
En el següent quadre podem veure com interactuaran els diferents components.

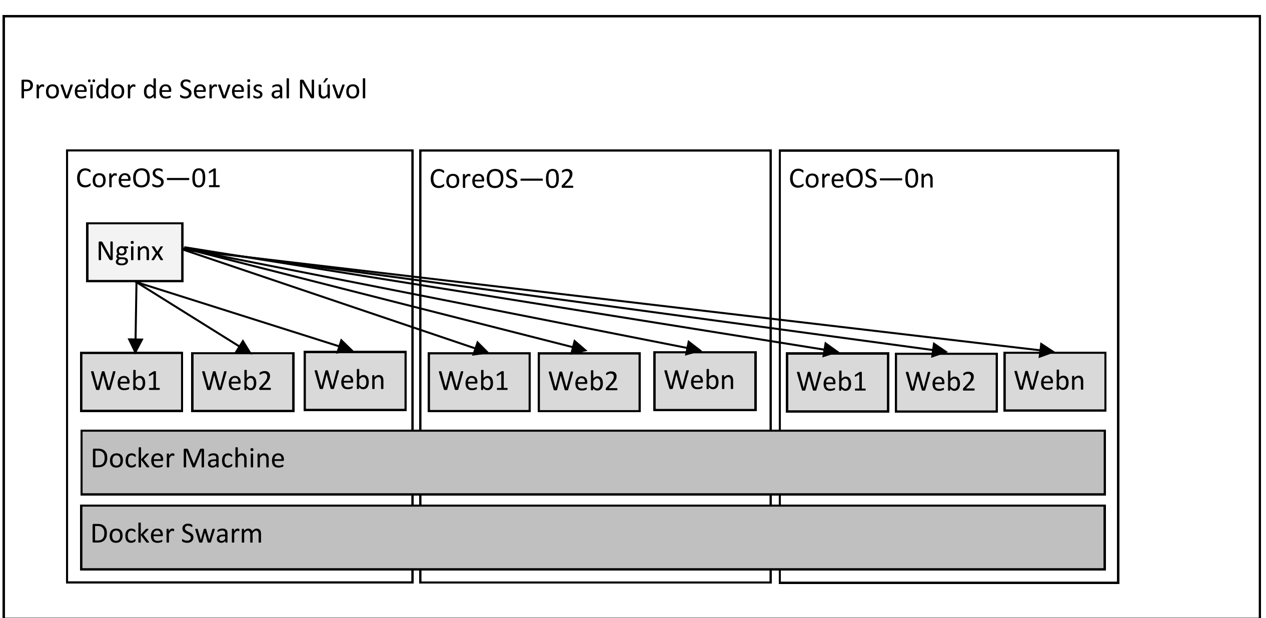

Figura 2 – Disseny Inicial

CoreOS[1]: https://coreos.com
Docker[2]: https://www.docker.com
Docker Swarm[3]: https://www.docker.com/docker-swarm
Docker Machine[4]: https://www.docker.com/docker-machine
Nginx[5]: https://www.nginx.com

Tindrem els servidors CoreOS necessaris per poder donar un servei òptim. Cada servidor tindrà un contenidor amb Docker Swarm i Docker Machine per tal de gestionar els diferents contenidors. Tindrem un únic contenidor amb Nginx que farà de proxy d’entrada i tants contenidors Web com necessitem. S’haurà d’estudiar quin és el numero òptim de contenidors web per servidor i en cas d’arribar al màxim haurem d’instanciar un nou servidor CoreOS per poder seguir desplegant més servidors Web.

Organització del Projecte

En aquest apartat, detallarem els recursos que caldrà assignar per aquest projecte.

Recursos Humans.

Es realitza la següent assignació de rols, què òbviament recauen sobre la mateixa persona, l'estudiant i autor del mateix:

Gerent del Projecte (GerPrj), Arquitecte de Maquinari (ArqMaq), Programador de Sistemes (ProSis) i Gestor de la Qualitat (GesQua).

Recursos Temporals.

Atès el caràcter docent del projecte, i el nombre de crèdits quadrimestrals universitaris assignats, el còmput global d'hores invertides no hauria de superar les 225.

Recursos Materials.

Documentals: Accés a la documentació online de totes les eines que es volen utilitzar.

Tecnològics: Accés i compte a DigitalOcean (proveïdor d'infraestructures en el núvol) per poder crear tots els servidors necessaris. I accés a internet ja que tot el projecte s'executarà al núvol.

Les eines ofimàtiques i l'accés consultori a Internet (que en general, no es consideraran fonts bibliogràfiques del projecte) no formaran part de recursos tecnològics consumits pel projecte atenent que avui dia, són les eines bàsiques de treball en qualsevol àmbit tecnològic.

La distribució d'aquests recursos, vindrà determina a l'apartat posterior, "Pla de Treball".

Pla de treball

Relació d'activitats

Enumerem les activitats i tasques que conformen el projecte.

1. Cerca d'informació del software que volem utilitzar.
2. Documentació del que volem aconseguir.
3. Disseny de l'arquitectura objectiu.
4. Instal·lació i configuració de tots els components.
    a. Instal·lació i configuració CoreOS.
    b. Instal·lació i configuració Docker.
    c. Instal·lació i configuració Nginx.
    d. Instal·lació i configuració Docker Swarm.
    e. Instal·lació i configuració Docker Machine.
5. Configuració del procés automàtic d'arrencada i parada de servidors.
6. Realitzar proves de tots els components per separat.
7. Realitzar proves dels components en conjunt.
8. Documentar els resultats de cadascuna de les proves.
9. Realitzar la memòria del projecte.

Fites principals

El gestor del projecte garantirà, en tot moment, que les fites principals quedin assolides en els termes i terminis d'aquesta proposta.

FITA	DESCRIPCIÓ
F1	Estudi i proposta de les tecnologies a utilitzar.
F2	Entrega del programari.
F3	Entrega de la memòria final del projecte.

Taula 2 - Fites del projecte

Calendari de treball

Ara establirem el calendari laboral per a l'execució del projecte. Aquest estarà format per blocs setmanals, de dilluns a divendres, exceptuant els dies festius. L' inici correspon al dilluns 21 de setembre de 2015 i l' acabament al divendres 22 de gener de 2016. Això comporta un total de

17 setmanes, on s’han d’extreure dos setmanes per festes de nadal i els dies 12 d’octubre i 7 i 8 de desembre. En total, es disposarà de 72 dies laborables destinats al projecte.

Quant a la franja horària laboral, es destina la corresponent a l’interval de 20 a 23h. Això fa un total de 3h diàries, amb un total de 225 hores.

Adjuntem a continuació un diagrama de Gantt, on es mostra, concurrentment en el temps, el decurs de totes les activitats del projecte fins el seu tancament.

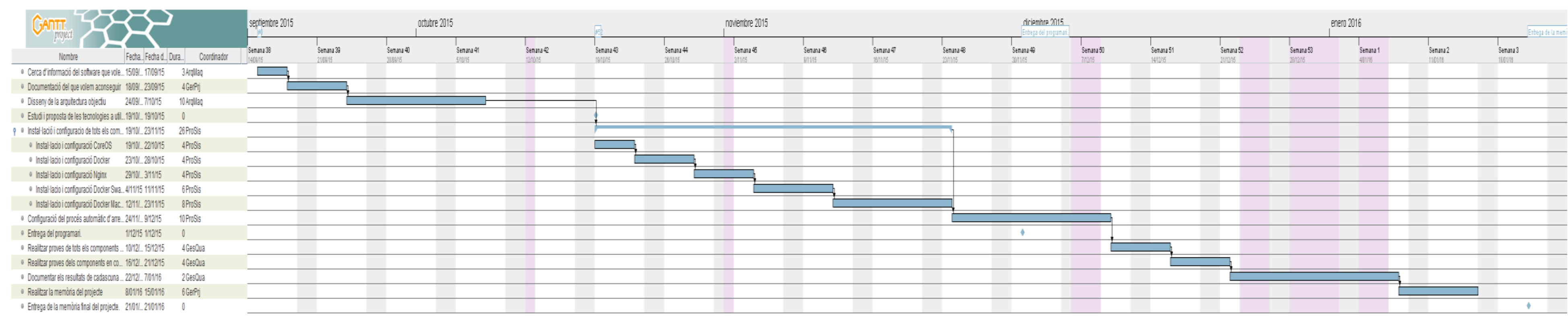

Figura 3 - Diagrama de Gantt

Valoració Econòmica

Finalment, adjuntem la informació de caràcter econòmic que engloba el projecte. Establint costos a cadascun dels recursos humans i materials assignats al projecte podem sintetitzar els resultats en les següents taules:

NOM RECURS	CODI	PREU (€/h)	JORNADES	HORES	COST
Gerent del Projecte	GerPrj	60	10	30	1800€
Arquitecte de Maquinari	ArqMaq	50	13	39	1950€
Programador de Sistemes	ProSis	40	36	108	4320€
Gestor de la Qualitat	GesQua	40	10	30	1200€

Taula 3 - Recursos

ID	ACTIVITAT	COST
1	Cerca d'informació del software que volem utilitzar.	450€
2	Documentació del que volem aconseguir.	720€
3	Disseny de l'arquitectura objectiu.	1500€
4	Instal·lació i configuració de tots els components.	3120€
4.1	Instal·lació i configuració CoreOS.	480€
4.2	Instal·lació i configuració Docker.	480€
4.3	Instal·lació i configuració Nginx.	480€
4.4	Instal·lació i configuració Docker Swarm.	720€
4.5	Instal·lació i configuració Docker Machine.	960€
5	Configuració del procés automàtic d'arrencada i parada de servidors.	1200€
6	Realitzar proves de tots els components per separat.	480€
7	Realitzar proves dels components en conjunt.	480€
8	Documentar els resultats de cadascuna de les proves.	240€
9	Realitzar la memòria del projecte.	1080€
	Total:	9270€

Taula 4 - Valoració activitats

Cost Material: Per realitzar el projecte tota la documentació la trobarem a internet. Només serà necessari tenir un entorn virtual on poder crear les màquines i veure que tot funciona correctament.

Per fer-ho hem escollit el proveïdor de serveis en el núvol Digital Ocean. Utilitzarem 3 màquines amb les següents característiques:

CARACTERÍSTICA	VALOR
Cpu	1 cpu
Memòria	512 Mb
Disc	20 Gb

Taula 5 - Característiques servidor

En aquest proveïdor el preu mensual de la màquina és 5€ al mes. Per tant contant que el projecte dura 4 mesos el cost en material del projecte serà el següent:

Servidor	Preu Mensual	Preu Total
CoreOS_01	5€	20€
CoreOS_02	5€	20€
CoreOS_03	5€	20€
	Total:	60€

Taula 6 - Valoració materials

Arquitectura

Un cop iniciat el projecte hem detectat que l'arquitectura proposada inicialment [(Figura 1 – Disseny Inicial)] no era la millor opció. Per tant s'ha modificat per intentar donar la millor resposta a les necessitats inicials.

El principal obstacle ha estat no poder utilitzar CoreOS per tots els nodes del clúster. El docker-machine encara no està preparat per poder desplegar nodes de CoreOS amb Swarm integrat. De moment aquesta integració només és possible utilitzant Ubuntu. Per tant, ens hem vist obligats a fer servir Ubuntu pel Swarm-Master i pels Swarm-Nodes.

Un altre punt important és que inicialment es va planificar desplegar el proxy NGINX dins del mateix node que fa de Control. Inicialment es va fer així però al realitzar les proves es va detectar que quan tenim una carga elevada de tràfic el mateix node que està rebent tot aquest tràfic és l'encarregat d'anar validant quin és l'estat dels clúster de Swarm i això provocava, en alguns casos, que la màquina es saturés. Per aquest motiu es va decidir dedicar un servidor virtual per desplegar Nginx i així alliberar de carga el servidor de control.

L'arquitectura resultant després d'aquest canvis és la següent:

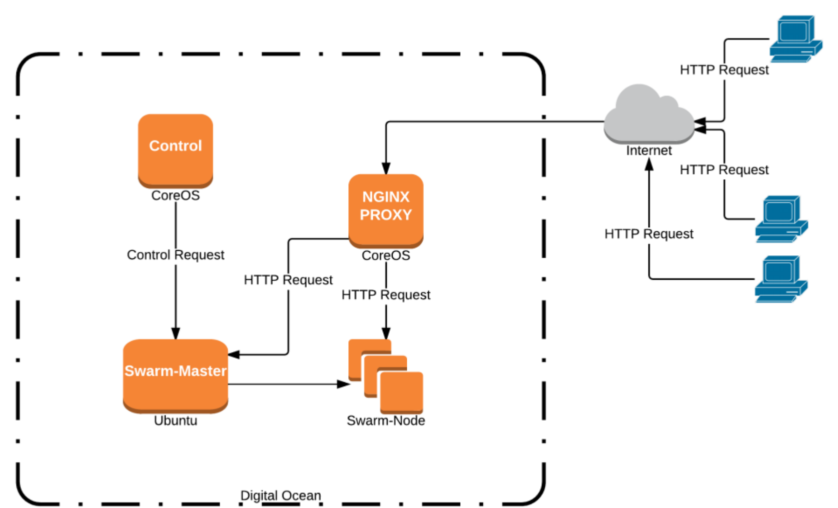

Figura 4 – Arquitectura Final

Com podem veure tenim dos nodes amb CoreOS que corresponen al servidor de control i al NGINX Proxy. El node on s'executa el Swarm-Master i els Swarm-Node utilitzen Ubuntu.

La funció del servidor de control és instanciar el Swarm-Master i després anar comprovant l'estat del clúster per veure si és necessari ampliar el nombre de servidors o disminuir-lo. La

base de l'arquitectura sempre és el servidor Swarm-Master amb dos contenidors que serveixen l'aplicació web. Si és necessari s'aniran instanciant nodes de swarm per poder anar desplegant més contenidors amb l'aplicació web. En cas que el tràfic disminueixi es destruiran però sempre com a mínim ens quedarem amb el Swarm Master i els seus dos contenidors.

Servidor de Control

El servidor de control és l'encarregat de gestionar el clúster. És el que executarà diferents scripts per dur a terme totes les tasques. La principal és controlar l'estat del clúster per decidir en quin moment s'ha d'ampliar o reduir.

Figura 5 – Servidor de Control

Per instal·lar aquest servidor en CoreOS necessitem definir un fitxer de configuració cloud-config.yaml. En aquest cas s'han definit els diferents serveis que volem instal·lar. Pel projecte necessitem instal·lar docker-machine, etcd i els script necessaris per controlar el clúster de contenidors.

```
#cloud-config
coreos:
  #Deshabilitem l'update del servidor automàtic
  update:
    reboot-strategy: off
  #Definim les variables del servei etcd2
  etcd2:
    # generate a new token for each unique cluster from https://discovery.etcd.io/new?size=1
    discovery: https://discovery.etcd.io/641d8a2fef59d18aa670fd0104bf99b4
    # multi-region and multi-cloud deployments need to use $public_ipv4
    advertise-client-urls: http://$public_ipv4:2379
    initial-advertise-peer-urls: http://$private_ipv4:2380
    # listen on both the official ports and the legacy ports
    # legacy ports can be omitted if your application doesn't depend on them
    listen-client-urls: http://0.0.0.0:2379,http://0.0.0.0:4001
    listen-peer-urls: http://$private_ipv4:2380
  units:
    #Definim el servei etcd2 perquè arrenqui a l'inici
    - name: etcd2.service
      command: start
    #Definim el servei docker-machine perquè s'instal·li i arrenqui a l'inici
    - name: docker-machine.service
      runtime: true
      command: start
      content: |
        [Unit]
        Description=Install Docker Machine

        [Service]
        ExecStart=/usr/bin/bash -c '/usr/bin/mkdir -p /opt/bin && /usr/bin/curl -sSL
https://github.com/docker/machine/releases/download/v0.5.0-rc3/docker-machine_linux-386.zip > /tmp/docker-
machine_linux-386.zip && sudo /usr/bin/unzip /tmp/docker-machine_linux-386.zip && sudo /usr/bin/mv docker-machine*
/opt/bin/ && chmod +x /opt/bin/docker-machine && sudo /usr/bin/chmod -R 777 /opt/bin'
    #Definim el servei scripts-control perquè s'instal·li de forma automàtica
    - name: scripts-control.service
      runtime: true
      command: start
      content: |
        [Unit]
        Description=Install Scripts Control

        [Service]
        ExecStart=/usr/bin/bash -c '/usr/bin/mkdir -p /opt/bin && cd /opt/bin && sudo /usr/bin/git clone
https://urik23@bitbucket.org/urik23/autoscalableswarm.git && sudo /usr/bin/mv
/opt/bin/autoscalableswarm/scripts_control/* /opt/bin/ && sudo /usr/bin/rm -R /opt/bin/autoscalableswarm/ && sudo
/usr/bin/chmod -R 777 /opt/bin'
```

Per realitzar totes aquestes tasques s'han creat uns processos que s'executen des del servidor. Aquests són Start Container, Start Server, Destroy Server i Check CPU. A continuació descriurem cadascun dels processos amb detall. El primer ens permet crear un nou contenidor.

Start Container

Aquest procés ens permet instanciar un nou contenidor. Primer comprovem que no hi hagi cap operació en curs i arrenquem el contenidor. Si no tenim prou espai en el clúster per poder-ho fer hem de crear un nou servidor. Quan el servidor estigui inclòs dins del clúster llavors ja podrem arrencar el contenidor. A continuació tenim el diagrama d'estats.

START CONTAINER

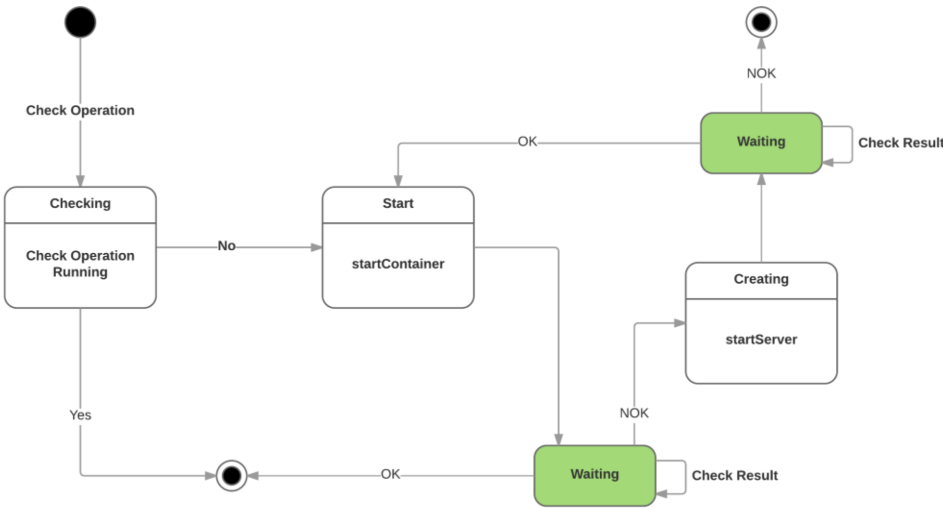

Figura 6 – Diagrama Estats Start Container

El codi és el següent:

```
#!/bin/bash
#Creem el fitxer de control
sudo touch /opt/bin/serverOperation

#Carreguem les variables d'entorn de dockerSwarm
eval "$(docker-machine env --swarm swarm-master)"

#Recuperem la variable d'entorn VHOSST per passar-la al nou contenidor que arrenquem
VHOST=`etcdctl get /GLOBAL/VHOST`

#Executem la comanda de docker per arrencar el nou contenidor
RESULTAT=$(docker run -d -p 8080 -m 200M --env VHOST=$VHOST urik/web-uoc)

#Si ens retorna un missatge és que no ha pogut crear el nou contenidor i necessitem un nou servidor
if [ -z "$RESULTAT" ];
then
  #Recuperem el nombre de nodes que tenim actualment
  nodesSwarmObj=$(docker info | grep Nodes | awk '{print $2}')
  nodesSwarm=$nodesSwarmObj
  #Arrenquem el contendior
  sh startServer.sh >> sudo /opt/bin/server.log &
  segons=0
  echo $nodesSwarmObj
  echo $nodesSwarm
      #Fem un bucle per esperar que tot el procés hagi acabat. Si tarda més de 300s ho donem com error.
      while [ "$nodesSwarm" -eq "$nodesSwarmObj" ] && [ $segons -lt 300 ]; do
         sleep 5
         ((segons=segons+5))
         #Recuperem el nombre de nodes per comprobar amb l'inicial.
         nodesSwarm=$(docker info | grep Nodes | awk '{print $2}')
         echo $segons "secs. -" $nodesSwarm "nodes swarm"
      done
  #Al sortir comprovem si ha estat perquè tot està correcte o perquè hi ha hagut un timeout.
  if [ "$nodesSwarm" -ne "$nodesSwarmObj" ]; then
    #Si tot ha estat correcte arrenquem de nou un contenidor perquè ara ja tindrem espai al clúster
    sh startContainer.sh >> sudo /opt/bin/server.log &
  fi
fi

#Eliminem el fitxer de control per poder realitzar més operacions
sudo rm -rf /opt/bin/serverOperation
```

Els punts més destacats són:

```
RESULTAT=$(docker run -d -p 8080 -m 200M --env VHOST=$VHOST urik/web-uoc)
```

Amb aquesta instrucció arrenquem un nou contenidor.

run: Crear i arrencar un nou contenidor

-d: En mode detach. Podem executar el contenidor en background

-p 8080: Exposem el port 8080 del contenidor amb un aleatori del servidor. D'aquesta manera podem arrencar més d'un contenidor al mateix servidor. Després serà el NGINX qui s'encarregarà de mapejar aquests ports en el proxy. En aquest cas tenim dos contenidors que es publiquen pels ports 32768 i 32769.

```
core@control ~ $ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
NAMES
90497c53bdd6            urik/web-uoc            "java  -Djava.securit    3  seconds  ago          Up  2  seconds
178.62.84.89:32769->8080/tcp    swarm-node-1/boring_sammet
009cbf80775e            urik/web-uoc            "java  -Djava.securit    5  minutes  ago          Up  5  minutes
178.62.84.89:32768->8080/tcp    swarm-node-1/grave_kowalevski
```

-m 200M: Assignem 200M del servidor al contenidor

--env VHOST=$VHOST: Creem al contenidor la variable d'entorn VHOST amb el valor $VHOST que en el nostre cas l'agafem de la variable /GLOBAL/VHOST que tenim a etcd amb la instrucció `` VHOST=`etcdctl get /GLOBAL/VHOST` ``

Amb aquesta instrucció recuperem el nombre de nodes del clúster.

```
nodesSwarmObj=$(docker info | grep Nodes | awk '{print $2}')
```

```
core@control ~ $ docker info
Containers: 1
Images: 1
Role: primary
Strategy: spread
Filters: health, port, dependency, affinity, constraint
Nodes: 1
 swarm-node-1: 178.62.84.89:2376
  └ Containers: 1
  └ Reserved CPUs: 0 / 1
  └ Reserved Memory: 0 B / 514.5 MiB
  └ Labels: executiondriver=native-0.2, kernelversion=3.13.0-57-generic, operatingsystem=Ubuntu 14.04.3 LTS,
provider=digitalocean, storagedriver=aufs
CPUs: 1
Total Memory: 514.5 MiB
Name: 471c915c4ec2
core@control ~ $ docker info | grep Nodes | awk '{print $2}'
1
core@control ~ $
```

A continucació definim el procés que ens permet crear i arrencar un nou servidor.

Start Server

Aquest procés ens permet arrencar un nou servidor. Com que podem tenir diferents nodes de swarm el que fem primer és recuperar el número de servidor que toca crear. Llavors executem la comanda de creació passant aquest número i si tot ha anat bé augmentem en un.

START SERVER

getNumServer

Getting

Get NumServer

Creating

Start Server

Refresh

numServer++

Figura 7 - Diagrama Estats Start Server

El codi és el següent:

```
#!/bin/bash

#Recuperem per quin node de swarm estem del etcd
NUM_SERVER=`etcdctl get /GLOBAL/SERVER_NUM`

#Recuperem el token de digital ocean de etcd
export DIGITAL_OCEAN_TOKEN=`etcdctl get /GLOBAL/DIGITAL_OCEAN_TOKEN`

#Recuperem el token de swarm del etcd
export SWARM_TOKEN=`etcdctl get /GLOBAL/SWARM_TOKEN`

#Executem la comanda de docker-machine per crear un nou servidor
docker-machine create -d digitalocean --digitalocean-region lon1 --digitalocean-size "512mb" --digitalocean-access-
token $DIGITAL_OCEAN_TOKEN --swarm --swarm-discovery $SWARM_TOKEN swarm-node-`etcdctl get /GLOBAL/SERVER_NUM`

#Augmentem en un el comptador de nodes de swarm
NUM_SERVER=`etcdctl get /GLOBAL/SERVER_NUM`
((NUM_SERVER++))
etcdctl set /GLOBAL/SERVER_NUM $NUM_SERVER
```

Els punts més destacats són:

```
docker-machine create -d digitalocean --digitalocean-region lon1 --digitalocean-size "512mb" --digitalocean-access-
token $DIGITAL_OCEAN_TOKEN --swarm --swarm-discovery $SWARM_TOKEN swarm-node-`etcdctl get /GLOBAL/SERVER_NUM`
```

Amb aquesta instrucció creem un nou servidor amb docker-machine a Digital Ocean

create: Crear un nou servidor

-d digitalocean: Indiquem el driver que volem utilitzar. En el nostre cas digitalocean però hi ha altres proveïdors.

--digitalocean-region lon1: En quina regió volem crear el servidor.

--digitalocean-size "512mb": La memòria que volem assignar

--digitalocean-access-token $DIGITAL_OCEAN_TOKEN: El token per poder accedir.

--swarm: Per indicar què volem utilitzar swarm

--swarm-discovery $SWARM_TOKEN: El token del nostre clúster de swarm

swarm-node-`etcdctl get /GLOBAL/SERVER_NUM`: El nom del nou node. En el nostres cas els anem enumerant de forma correlativa amb la variable SERVER_NUM.

A continuació definim el procés que ens permetrà eliminar servidors del clúster sempre que sigui necessari.

Destroy Server

Aquest procés ens permet eliminar un servidor del clúster. Abans d'eliminar el servidor l'hem de parar. Si el nombre de servidor és igual a 1 no l'eliminem ja que vol dir que estem només amb el swarm master i sempre ha d'estar actiu amb els seu contenidors. Un cop eliminat el servidor fem decréixer en un el comptador de servidors.

DESTROY SERVER

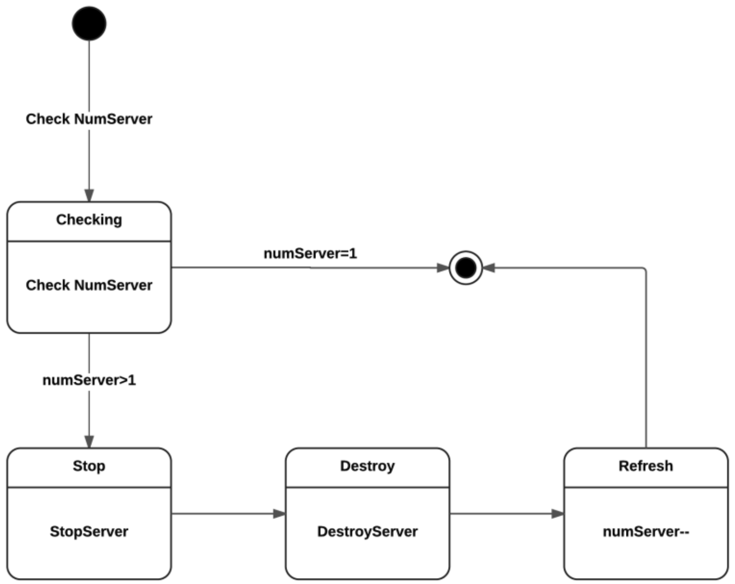

Figura 8 - Diagrama Estats Destroy Server

El codi és el següent:

```
#!/bin/bash

#Recuperem el número del servidor a eliminar
NUM_SERVER=`etcdctl get /GLOBAL/SERVER_NUM`

#Si és igual a 1 vol dir que ja estem en el mínim i no podem eliminar.
if [ ! "$NUM_SERVER" -eq 1 ]; then

#Disminuïm en 1 el número de servidor i ho guardem a etcd
((NUM_SERVER--))
etcdctl set /GLOBAL/SERVER_NUM $NUM_SERVER

#Primer parem el servidor i després l'eliminem.
docker-machine stop swarm-node-`etcdctl get /GLOBAL/SERVER_NUM`
docker-machine rm swarm-node-`etcdctl get /GLOBAL/SERVER_NUM`

fi
```

Els punts més destacats són:

```
docker-machine stop swarm-node-`etcdctl get /GLOBAL/SERVER_NUM`
docker-machine rm swarm-node-`etcdctl get /GLOBAL/SERVER_NUM`
```

Amb aquestes instruccions parem i eliminem el servidor de digital ocean.

stop: Per parar el servidor

rm: Per eliminar el servidor

A continuació definim el procés que monitoritza la CPU del clúster per saber quan és necessàri destinar més recursos.

Check CPU

Aquest procés és l'encarregat d'anar controlant la mitja de CPU del clúster i en cas de passar uns umbrals definits crear un nou contenidor o servidor. En canvi si detecta que la CPU baixa per sota de l'umbral definit elimina el servidor. En definitiva, és l'encarregat de mantenir el clúster amb els nodes necessaris en tot moment sempre mantenint com a mínim el servidor del swarm master.

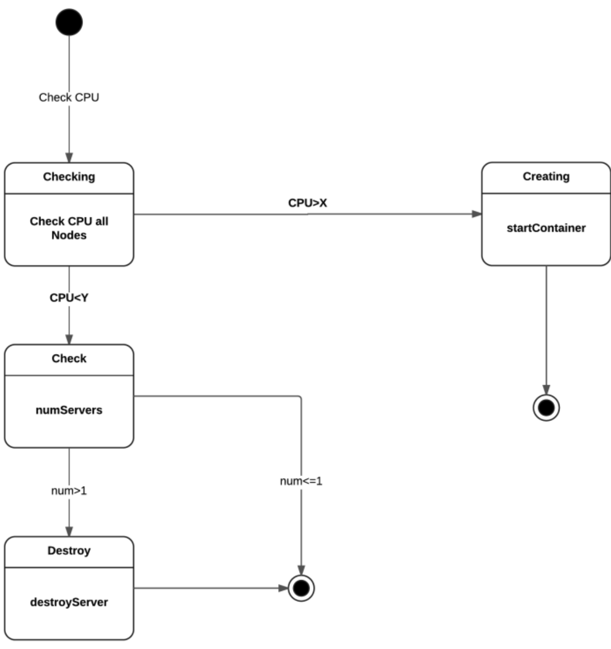

Figura 9 - Diagrama Estats Check CPU

El codi és el següent:

```
#!/bin/bash

#Comprobem que no hi hagi una operació en execució.
if [ ! -e /opt/bin/serverOperation ]; then

#Carreguem les variables d'entorn per swarm
eval "$(docker-machine env --swarm swarm-master)"

  #Si no existeix creem el fitxer acuMitja on guardarem les últimes 5 mitjanes de CPU
```

```
  sudo touch /opt/bin/acuMitja
  sudo chmod 777 /opt/bin/acuMitja

  #Guardem a la variable lin el número de línies que té el fitxer acuMitja
  lin=$(sudo wc -l < /opt/bin/acuMitja)

  #Si és igual a 5 eliminem la primera i ens quedem amb les 4 últimes.
  if [ "$lin" -eq 5 ]; then
     sudo tail --lines=4 /opt/bin/acuMitja > /opt/bin/acuMitjaTemp
     sudo mv /opt/bin/acuMitjaTemp /opt/bin/acuMitja
  fi

  sudo chmod 777 /opt/bin/acuMitja

  #Per cada contenidor que tenim en execució calculem la mitja i afegim el resultat a acuMitja.
  docker ps -q | awk -f /opt/bin/cpu.awk -v lin=$lin >> /opt/bin/acuMitja

  #Calculem la mitja dels valors que tenim acuMitja i ho guardem a la variable mitja
  mitja=$(awk '{total+=$1} END{print total/NR}' /opt/bin/acuMitja)

  #Recuperem el valor màxim de cores i el comparem amb la mitja
  echo $mitja
  max_core=`etcdctl get /GLOBAL/MAX_CPU_CORE`
  var=$(awk -v mitja=$mitja -v max_core=$max_core 'BEGIN{ if ( mitja > max_core ) {print 1} else {print 0} }')

  #Si és superior arrenquem un nou contenidor perquè necessitem més recursos.
  echo "var1 " $var
  if [ "$var" -eq 1 ]; then
    echo "StartContainer"
    sh startContainer.sh >> sudo /opt/bin/server.log &
  fi

  #Recuperem el valor mínim de cores i el comparem amb la mitja.
  min_core=`etcdctl get /GLOBAL/MIN_CPU_CORE`
  var=$(awk -v mitja=$mitja -v min_core=$min_core 'BEGIN{ if ( mitja < min_core ) {print 1} else {print 0} }')

  #Si és inferior eliminem un servidor perquè no necesitem tants recursos.
  echo "var2 " $var
  if [ "$var" -eq 1 ]; then
    serverNum=`etcdctl get /GLOBAL/SERVER_NUM`
    echo "Comprobem quants servidors tenim: " $serverNum
    if [ "$serverNum" -gt "1" ]; then
      echo "Destruim Servidor"
      sh destroyServer.sh >> sudo /opt/bin/server.log &
    fi
  fi
fi
```

Els punts més destacats són:

```
docker ps -q | awk -f /opt/bin/cpu.awk -v lin=$lin >> /opt/bin/acuMitja
```

El primer paràmetre ens retorna l' id de tots els contenidors que estem executant

```
core@control ~/autoscalableswarm $ docker ps
CONTAINER ID        IMAGE               COMMAND                CREATED             STATUS              PORTS
NAMES
90497c53bdd6        urik/web-uoc        "java -Djava.securit   3 hours ago         Up 3 hours
178.62.84.89:32769->8080/tcp   swarm-node-1/boring_sammet
804dd3481605        urik/web-uoc        "java -Djava.securit   3 hours ago         Up 3 hours
46.101.48.131:32768->8080/tcp   swarm-master/drunk_saha
009cbf80775e        urik/web-uoc        "java -Djava.securit   3 hours ago         Up 3 hours
178.62.84.89:32768->8080/tcp   swarm-node-1/grave_kowalevski
core@control ~/autoscalableswarm $ docker ps -q
90497c53bdd6
804dd3481605
009cbf80775e
core@control ~/autoscalableswarm $
```

El segon paràmetre agafa l'entrada del primer i executa el codi de cpu.awk i deixa el resultat concatenat al fitxer acuMitja.

```
#Només una vegada al principi inicialitzem la variable a
BEGIN       {
```

```
            a = 0  # En realitat, no cal, perque ja s'inicialitza a zero.
            }
            #Per cada valor de l'entrada executem el script extractCpu.sh passant l'id del contenidor
            {cmd = "extractCpu.sh " $1
             #El resultat el guardem a la variable var
             while ((cmd | getline var) > 0)
                {
                 # printf("var = %s\n", var)      # No cal, és només per debug.
                #Anem acumulant a la variable a la suma dels consums de cpu de cada contenidor
                a += var
                }
             close(cmd)
            }

END         {
             #print a     # No cal, és només per debug.
            #Només una vegada al final retornem la suma de totes les cpus divit pel número de registres
             #La mitjana de la cpu de tots els contenidors que tenim en execució
             print a / NR
             }
```

Per extreure la CPU de cada contenidor fem servir el script extracCpu.

```
#!/bin/bash

#Carreguem les variable per swarm
eval "$(docker-machine env --swarm swarm-master)"

#Executem la crida a la funció metrics del contenidor i ens quedem amb la cpu.
docker exec --interactive=false $1 curl -s http://localhost:8080/metrics | sed
's/^.*systemload.average":\(.*\),"heap.committed".*/\1/'
```

El punt més destacat és:

```
docker exec --interactive=false $1 curl -s http://localhost:8080/metrics | sed
's/^.*systemload.average":\(.*\),"heap.committed".*/\1/'
```

Amb el primer paràmetre recuperem les mètriques del contenidor

exec: Per executar una comanda

--interactive=false: S'executa la comanda i es retorna el control

$1: Passem el id del contenidor on volem executar la comanda

curl -s http://localhost:8080/metrics: La comanda a executar

```
core@control ~/autoscalableswarm $ docker ps -q
90497c53bdd6
804dd3481605
009cbf80775e
core@control ~/autoscalableswarm $ docker exec --interactive=false 90497c53bdd6 curl -s http://localhost:8080/metrics
{"mem":34996,"mem.free":4542,"processors":1,"instance.uptime":12684895,"uptime":12699382,"systemload.average":0.0,"hea
p.committed":34996,"heap.init":8192,"heap.used":30453,"heap":124736,"threads.peak":12,"threads.daemon":8,"threads":10,
"classes":5348,"classes.loaded":5348,"classes.unloaded":0,"gc.copy.count":156,"gc.copy.time":610,"gc.marksweepcompact.
count":3,"gc.marksweepcompact.time":203,"httpsessions.max":-1,"httpsessions.active":0}
```

Amb el segon paràmetre agafem el resultat del primer i ens quedem només amb el valor de la CPU que està identificat per el systemload.average.

```
sed 's/^.*systemload.average":\(.*\),"heap.committed".*/\1/'
```

sed: La instrucció
s: Substituir
/: Delimitador
^: Des del principi
*.*systemload.average"::* Qualsevol caràcter fins a systemload.average":
\(.\),"heap.committed".**: Guardem com una variable fins a "heap.committed" i fins al final
/\1/: Substituïm tot pel que ens hem guardat

Tot el codi del servidor de control el podem trobar en el repositori de git https://bitbucket.org/urik23/autoscalableswarm.git dins la carpeta scripts_control.

A continuació definim l'arquitectura utilitzada per crear el NGINX Proxy.

NGINX Proxy

El servidor NGINX Proxy és l'encarregat de distribuir les peticions que ens arriben entre tots els contenidors del clúster. Per tant ha de ser capaç, d'una forma dinàmica, d'anar actualitzant la seva configuració per afegir o treure cada contenidor que parem o arranquem. D'aquesta manera, quan detectem que la CPU del sistema augmenta i el servidor de control arrenqui més contenidors el NGINX Proxy els afegirà a la seva configuració i de forma transparent per l'usuari començarà a enviar peticions a aquests nous recursos aconseguint així que la CPU disminueixi. De la mateixa manera quan es pari i s'elimini algun contenidor haurem de treure'l de la configuració perquè no hi enviem cap més petició.

Per poder fer aquest procés de forma transparent farem que el servidor NGINX escolti al end point del servidor swarm master per detectar qualsevol canvi i poder actualitzar el seu fitxer de configuració.

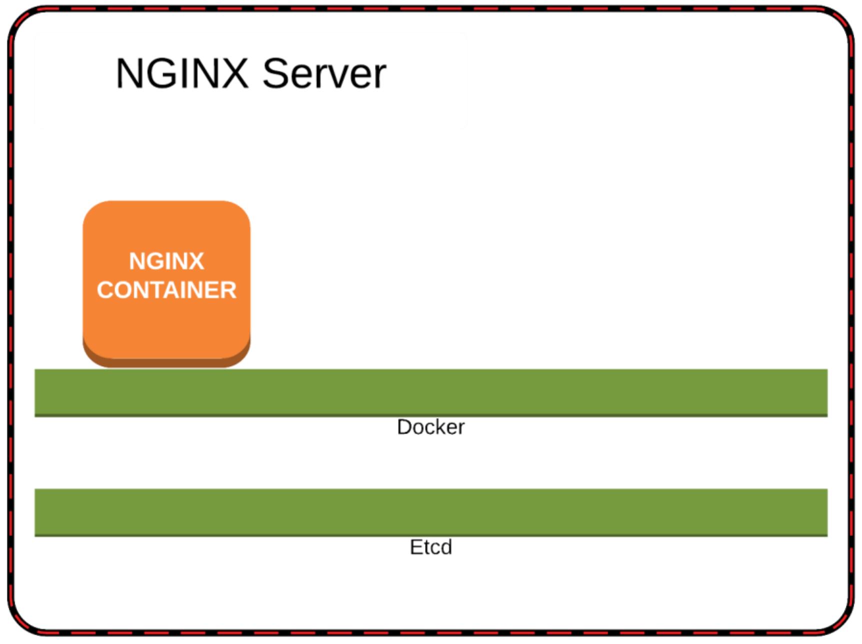

Figura 10 – NGINX Server

El primer que necessitem és crear un contenidor on instal·larem el NGINX i les eines necessàries per poder detectar els canvis en el clúster de swarm. Per crear un contenidor el fitxer principal que necessitem és un Dockerfile.

```
#Indiquem de quina distribució volem partir
FROM nginx:1.9.5
MAINTAINER Oriol Moya Ahufinger oriol.moya@gmail.com
#Creem els directoris que necessitem
#Creació de directoris
RUN mkdir -p /opt/bin
RUN mkdir -p /certs
RUN mkdir -p /template
#Instal·lem el wget per poder descarregar el docker-gen
#Instal·lar wget
RUN apt-get update \
 && apt-get install -y -q --no-install-recommends \
    wget
#Instal·lem el docker-gen per poder capturar els events d'arrencada i parada dels dockers i modificar amb un
#template la configuració del Nginx.
#Instal·lar docker-gen
RUN wget https://github.com/jwilder/docker-gen/releases/download/0.4.3/docker-gen-linux-amd64-0.4.3.tar.gz
RUN tar xvzf docker-gen-linux-amd64-0.4.3.tar.gz
RUN chmod +x docker-gen
RUN mv docker-gen /opt/bin
#Copiem el template, la configuració i el script d'inici.
#Copiar template, conf i script start
```

```
ADD assets/nginx.tmpl /template/nginx.tmpl
ADD assets/nginx.conf /etc/nginx/nginx.conf
ADD assets/run.sh /opt/bin/run.sh
#Donem permís d'execució
RUN chmod +x /opt/bin/run.sh
#Fitxer que executem a l'arrencar el contenidor
CMD ["/opt/bin/run.sh"]
```

A part del Dockerfile tenim els següents fitxers dins la carpeta assets i que seran utilitzats pel Dockerfile. Per poder actualitzar de forma dinàmica el servidor de NGINX fem servir docker-gen que ens permet detectar els canvis i a partir d'un template refer un fitxer i executar una comanda. D'aquesta manera aconseguim que quan es para o s'engega un contenidor recuperem els valors, refem el fitxer de configuració i fem un reload de nginx perquè carregui aquesta nova configuració. El primer fitxer que necessitem és el nginx.tmpl que és el template que farem servir per crear el fitxer de configuració de NGINX.

```
daemon off;
worker_processes 1;
events { worker_connections 1024; }
http {
    sendfile on;
    gzip              on;
    gzip_http_version 1.0;
    gzip_proxied      any;
    gzip_min_length   500;
    gzip_disable      "MSIE [1-6]\.";
    gzip_types        text/plain text/xml text/css
                      text/comma-separated-values
                      text/javascript
                      application/x-javascript
                      application/atom+xml;

#En aquesta part agafem els valors que ens proporcionen els contenidors per poder crear el fitxer de
#configuració.
#Per cada servidor amb la mateixa variable VHOST s'escriu la seva ip i port dins de l'apartat upstream.
{{ range $host, $containers := groupByMulti $ "Env.VHOST" "," }}
upstream {{ $host }} {
  {{ range $index, $value := $containers }}{{ with $address := index $value.Addresses 0 }}server {{
$value.Node.Address.IP }}:{{ $address.HostPort }};
  {{ end }}{{ end }}
}
server {
  gzip_types text/plain text/css application/json application/x-javascript text/xml application/xml
application/xml+rss text/javascript;
#Definim el nom del domini
server_name {{ $host }};
  proxy_buffering off;
  error_log /proc/self/fd/2;
  access_log /proc/self/fd/1;
```

```
    location / {
      #Indiquem que qualsevol petició s'ha de distribuïr entre els servidors definits dins del upstream amb el
  #mateix nom que el proxy pass.
      proxy_pass http://{{ $host }};
      proxy_set_header Host $http_host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Proto $scheme;

      # HTTP 1.1 support
      proxy_http_version 1.1;
      proxy_set_header Connection "";
      }
    }
  {{ end }}
```

En el fitxer run.sh definim el que executarem quan iniciem un contenidor. En el nostre cas el nginx i el docker-gen que ens actualitzarà la configuració.

```
#!/bin/bash

#Executem docker-gen en background
nohup /opt/bin/docker-gen -watch -only-exposed --tlscacert=/certs/ca.pem --tlscert=/certs/cert.pem --
tlskey=/certs/key.pem -endpoint=$ENDPOINT_SWARM_MASTER -notify "service nginx reload" /template/nginx.tmpl
/etc/nginx/nginx.conf &

#Executem nginx
nginx
```

El punt més destacat és:

```
nohup /opt/bin/docker-gen -watch -only-exposed --tlscacert=/certs/ca.pem --tlscert=/certs/cert.pem --
tlskey=/certs/key.pem -endpoint=$ENDPOINT_SWARM_MASTER -notify "service nginx reload" /template/nginx.tmpl
/etc/nginx/nginx.conf &
```

Amb aquesta instrucció arranquem docker-gen perquè escolti si hi ha canvis en els contenidors i actualitzi el fitxer de configuració.

-*watch*: que sempre estigui escoltant.

-*only-exposed*: Només els contenidors que tinguin ports exposats a l'exterior.

--*tlscacert=/certs/ca.pem*: El CA del swarm-master per poder accedir als events del docker del clúster de swarm.

--*tlscert=/certs/cert.pem*: El certificat del swarm-master per poder accedir als events del docker del clúster de swarm.

--*tlskey=/certs/key.pem*: La clau del swarm-master per poder accedir als events del docker del clúster de swarm.

-endpoint=$ENDPOINT_SWARM_MASTER: On ens hem de connectar per recuperar els events de Docker. Per exemple: tcp://104.236.2.3:3376

-notify "service nginx reload": Indiquem el que volem executar cada cop que rebem un event. En aquest cas recarreguem la configuració de nginx amb un reload.

Els dos últims valors són el template que volem fer servir (explicat en el punt anterior) i el path on volem deixar el template generat.

Per executar aquest contenidor farem la comanda:

```
docker run -d --name nginx-proxy -v /home/core/.docker/machine/certs:/certs/ -p 80:80 --env ENDPOINT_SWARM_MASTER=tcp://104.236.2.3:3376 urik/nginx-proxy-swarm
```

On la ip del endpoint ha de ser la del servidor swarm-master.

Un cop tenim definits tots els fitxers hem de crear la imatge a partir del Dockerfile. Per fer-ho executem la següent comanda.

```
docker build -t urik/nginx-proxy-swarm .
```

build: Per construir la imatge

-t urik/web-uoc: Nom de la imatge

. : Destinació local

```
core@control ~/autoscalableswarm/nginx-proxy-swarm $ docker build -t urik/nginx-proxy-swarm .
Sending build context to Docker daemon 10.24 kB
Sending build context to Docker daemon
Successfully built cb7f7245d9d1
```

Un cop creada la imatge la tenim en el servidor local on s'ha creat. Ho podem consultar amb la instrucció docker images.

```
core@control ~/autoscalableswarm/web-uoc $ docker images
REPOSITORY               TAG          IMAGE ID          CREATED          VIRTUAL SIZE
urik/web-uoc             latest       d7e1c2ffd4b7      4 minutes ago    667.4 MB
urik/nginx-proxy-swarm   latest       cb7f7245d9d1      33 minutes ago   198.9 MB
java                     8            6705ebcea7a3      3 days ago       641.9 MB
nginx                    1.9.5        3669a1f15a6f      4 weeks ago      132.7 MB
```

Per poder-la utilitzar des de qualsevol servidor s'ha de pujar a un repositori d'imatges. Per aquest projecte hem utilitzat docker hub. Per pujar l'imatge executem docker push.

```
core@control ~/autoscalableswarm/nginx-proxy-swarm $ docker push urik/nginx-proxy-swarm
The push refers to a repository [urik/nginx-proxy-swarm] (len: 1)
cb7f7245d9d1: Image push failed
```

```
Please login prior to push:
Username: urik
Password:
Email: oriol.moya@gmail.com
WARNING: login credentials saved in /home/core/.docker/config.json
Login Succeeded
The push refers to a repository [urik/nginx-proxy-swarm] (len: 1)
cb7f7245d9d1: Image already exists
3e7ce232c8b9: Image successfully pushed
fa24e7027e3e: Image successfully pushed
a528697661f1: Image successfully pushed
ae6c1bcfb6db: Image successfully pushed
7b587a0403df: Image successfully pushed
b1a01d74bfe4: Image successfully pushed
6419da60ac4d: Image successfully pushed
47487a78cbdf: Image successfully pushed
07b5eedd4306: Image successfully pushed
3ec74dd1dea7: Image successfully pushed
42116abf6dd6: Image successfully pushed
207a81d47bd5: Image successfully pushed
6ca38d803cb4: Image successfully pushed
3669a1f15a6f: Image already exists
c1ff64fa1aa6: Image already exists
5cd1c00ad84f: Image already exists
e240c7325698: Image successfully pushed
738e7a471e87: Image successfully pushed
5ac100425925: Image successfully pushed
fadf699bccc0: Image already exists
98de8af78960: Image successfully pushed
77f88505fa0d: Image successfully pushed
d0ea6b0b9e6a: Image already exists
6845b83c79fb: Image already exists
575489a51992: Image already exists

Digest: sha256:afbbbb6f7ba7be343c38c243ee42353a12bc70e5646dbc51032367dfa7794bca
```

Ara ja podem executar un contenidor amb aquesta imatge des de qualsevol servidor. Per fer-ho executem:

```
docker run -d --name nginx-proxy -v /home/core/.docker/machine/certs:/certs/ -p 80:80 --env
ENDPOINT_SWARM_MASTER=tcp://46.101.58.219:3376 urik/nginx-proxy-swarm
```

Si encara no tenim la imatge en local la descarregarà del repositori i si ja la tenim arrencarà directament. També podem baixar una imatge abans d'executar-la amb la següent comanda.

```
docker pull urik/nginx-proxy-swarm
```

En aquest cas com que ja la tenim creada en local no es baixa res.

```
core@control ~/autoscalableswarm/nginx-proxy-swarm $ docker push urik/nginx-proxy-swarm
The push refers to a repository [urik/nginx-proxy-swarm] (len: 1)
cb7f7245d9d1: Image already exists
3e7ce232c8b9: Image already exists
fa24e7027e3e: Image already exists
a528697661f1: Image already exists
ae6c1bcfb6db: Image already exists
7b587a0403df: Image already exists
b1a01d74bfe4: Image already exists
6419da60ac4d: Image already exists
47487a78cbdf: Image already exists
07b5eedd4306: Image already exists
3ec74dd1dea7: Image already exists
42116abf6dd6: Image already exists
207a81d47bd5: Image already exists
6ca38d803cb4: Image already exists
3669a1f15a6f: Image already exists
c1ff64fa1aa6: Image already exists
5cd1c00ad84f: Image already exists
e240c7325698: Image already exists
738e7a471e87: Image already exists
5ac100425925: Image already exists
fadf699bccc0: Image already exists
98de8af78960: Image already exists
77f88505fa0d: Image already exists
d0ea6b0b9e6a: Image already exists
6845b83c79fb: Image already exists
575489a51992: Image already exists
Digest: sha256:f03337768d416ad110c44a7828a396994be9615cade71cbe27f6d622ff8fb8e8
```

Tot el codi del NGINX Proxy el podem trobar en el repositori de git https://bitbucket.org/urik23/autoscalableswarm.git dins la carpeta nginx-proxy-swarm.

A continuació definim el contenidor que fem servir per desplegar l'aplicació Web.

Web UOC Container

Per poder fer les proves de tot el sistema s'ha creat un contenidor amb una aplicació web. La farem servir per enviar peticions a través de jMeter i provocar que la CPU augmenti per comprovar tot el sistema. Per aquest motiu, hem creat diferents serveis entre ells el càlcul de la sèrie de Fibonacci per provocar el consum de CPU.

Per crear el contenidor hem fet servir el següent Dockerfile

```
#Utilitzem com a base un contenidor amb java versió 8
FROM java:8
VOLUME /tmp
#Copiem el jar de l'aplicació
ADD wsCalculator-0.0.1-SNAPSHOT.jar app.jar
RUN bash -c 'touch /app.jar'
#Exposem el port 8080 per pdoer accedir a la aplicació
EXPOSE 8080
#Executem el jar al iniciar el contenidor.
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

En aquesta aplicació hem publicat diferents serveis que ens permeten calcular el nombre pi amb els decimals que indiquem o fer una seqüència de Fibonacci dels valors que vulguem.

Els més destacats són:

http://localhost/ - Retorna una pàgina estàtica amb el logo de la UOC, el text Hello World i el id del contenidor que ho està executant.

http://localhost/fibonacci/X - Retorna el resultat de calcular una seqüència de Fibonacci de X nombres.

http://localhost/pi/X -Retorna el resultat de calcular el nombre PI amb X decimals.

http://localhost/env - Retorna les variables d'entorn del sistema.

http://localhost/metrics - Retorna les mètriques del servidor. Aquest és el mètode que fem servir per calcular l'ús de CPU en el clúster i que ja hem comentat en l'apartat anterior.

http://localhost/trace - Retorna les últimes crides que s'han fet al sistema.

L'aplicació java està creada amb Spring Boot i porta el servidor tomcat incorporat per poder-lo executar directament.

El codi java més rellevant d'aquesta aplicació és el següent.

El fitxer App.java és el que arrenca l'aplicació.

```
package com.omoya.wsCalculator;

import org.springframework.boot.*;
import org.springframework.boot.autoconfigure.*;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.context.web.SpringBootServletInitializer;
```

```java
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan
@EnableAutoConfiguration
public class App extends SpringBootServletInitializer {

            public static void main(final String[] args) {
        SpringApplication.run(App.class, args);
    }

    @Override
    protected final SpringApplicationBuilder configure(final SpringApplicationBuilder application) {
        return application.sources(App.class);
    }
}
```

El fitxer Endpoint és on hem definit tots els mètodes que podem cridar.

```java
package com.omoya.wsCalculator;


import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;


@RestController
public class Endpoint {

            @RequestMapping(value = "/")
    public String hello() {
                        return generateWeb("hello", "world");
            }

            @RequestMapping(value = "/fibonacci/{numberStr}")
    public String fibonacci(@PathVariable String numberStr) {
                        int number = Integer.parseInt(numberStr);
                        String result = "";
                        for (int i = 1; i <= number; i++) {
                                    result = result + fib(i) + "\n";
                        }
                        return generateWeb("fibonacci", result);
    }

            @RequestMapping(value = "/pi/{numberStr}")
    public String pi(@PathVariable String numberStr) {
        return generateWeb("pi", pi_digits(Integer.parseInt(numberStr)));
    }

            private static long fib(int n) {
                        if (n <= 1)
                                    return n;
                        else
                                    return fib(n - 1) + fib(n - 2);
            }

            private static final int SCALE = 10000;
    private static final int ARRINIT = 2000;

    private static String pi_digits(int digits){
        StringBuffer pi = new StringBuffer();
        int[] arr = new int[digits + 1];
        int carry = 0;

        for (int i = 0; i <= digits; ++i)
            arr[i] = ARRINIT;

        for (int i = digits; i > 0; i-= 14) {
            int sum = 0;
            for (int j = i; j > 0; --j) {
                sum = sum * j + SCALE * arr[j];
                arr[j] = sum%(j * 2 - 1);
                sum /= j * 2 - 1;
            }

            pi.append(String.format("%04d", carry + sum / SCALE));
            carry = sum%SCALE;
        }
        return pi.toString();
    }

    private String generateWeb(String methodCalled, String resultToAdd) {
            String html = ""+ "\n";
            html = html + "<html>"+ "\n";
```

```
            html = html + "<head>"+ "\n";
            html = html + "<title>Test UOC PFC</title>"+ "\n";
            html = html + "<link href='http://fonts.googleapis.com/css?family=Open+Sans:400,700' rel='stylesheet' 
type='text/css'>"+ "\n";
            html = html + "<style>"+ "\n";
            html = html + "body {"+ "\n";
            html = html + "background-color: white;" + "\n";
            html = html + "text-align: center;" + "\n";
            html = html + "padding: 50px;" + "\n";
            html = html + "font-family: \"Open Sans\",\"Helvetica Neue\",Helvetica,Arial,sans-serif;" + "\n";
            html = html + "}"+ "\n";
            html = html + "#logo {"+ "\n";
            html = html + "margin-bottom: 40px;" + "\n";
            html = html + "}"+ "\n";
            html = html + "</style>"+ "\n";
            html = html + "</head>"+ "\n";
            html = html + "<body>"+ "\n";
            html = html + "<img id=\"logo\" 
src=\"http://www.uoc.edu/portal/_resources/common/imatges/marca_UOC/marca_UOC_blau_paper.png\" />"+ "<br>\n";
            html = html + "Servidor: " + System.getenv("HOSTNAME")+ "<br>\n";
            html = html + "Method called: " + methodCalled + "<br>\n";
            html = html + "Result: " + resultToAdd + "\n";
            html = html + "</body>"+ "\n";
            html = html + "</html>"+ "\n";
            return html;
    }

}
```

Per crear una imatge a partir del Dockerfile hem d'executar la següent instrucció.

```
docker build -t urik/web-uoc .
```

build: Per construir la imatge

-t urik/web-uoc: Nom de la imatge

. : Destinació local

```
core@control ~/autoscalableswarm/web-uoc $ docker build -t urik/web-uoc .
Sending build context to Docker daemon 12.83 MB
Sending build context to Docker daemon
Step 0 : FROM java:8
8: Pulling from java
5679b9b90e09: Pull complete
ea6bab360f56: Pull complete
0f062bc85662: Pull complete
1db7de304924: Pull complete
c8aab46f49f7: Pull complete
181e8ba79abb: Pull complete
d74c94329c2b: Pull complete
bae2b1c607c7: Pull complete
b30cb5f1d088: Pull complete
6796b7682b3e: Pull complete
d80ddc7f43a1: Pull complete
6705ebcea7a3: Pull complete
Digest: sha256:a29cc1f35faa99cb3b1177df21dc05f2f3ecb1ad12b90c96e34a32e1661a6a0c
Status: Downloaded newer image for java:8
 ---> 6705ebcea7a3
Step 1 : VOLUME /tmp
 ---> Running in 5bdb28ec3e11
 ---> de401b043452
Removing intermediate container 5bdb28ec3e11
Step 2 : ADD wsCalculator-0.0.1-SNAPSHOT.jar app.jar
 ---> 44df5ce8083a
Removing intermediate container 3a8742174d44
Step 3 : RUN bash -c 'touch /app.jar'
 ---> Running in 66e36c25ceb4
 ---> 5e8c2d826b70
Removing intermediate container 66e36c25ceb4
Step 4 : EXPOSE 8080
 ---> Running in d894349d8f5a
 ---> 6371d1f8981b
Removing intermediate container d894349d8f5a
Step 5 : ENTRYPOINT java -Djava.security.egd=file:/dev/./urandom -jar /app.jar
 ---> Running in 38d3da779a76
 ---> d7e1c2ffd4b7
Removing intermediate container 38d3da779a76
Successfully built d7e1c2ffd4b7
```

Un cop creada la imatge la tenim en el servidor local on s'ha creat. Ho podem consultar amb la instrucció docker images.

```
core@control ~/autoscalableswarm/web-uoc $ docker images
REPOSITORY               TAG                IMAGE ID           CREATED            VIRTUAL SIZE
urik/web-uoc             latest             d7e1c2ffd4b7       4 minutes ago      667.4 MB
urik/nginx-proxy-swarm   latest             cb7f7245d9d1       33 minutes ago     198.9 MB
java                     8                  6705ebcea7a3       3 days ago         641.9 MB
nginx                    1.9.5              3669a1f15a6f       4 weeks ago        132.7 MB
```

Per poder-la utilitzar des de qualsevol servidor s'ha de pujar a un repositori d'imatges. Per aquest projecte hem utilitzat docker hub. Per pujar la imatge executem docker push.

```
core@control ~/autoscalableswarm/web-uoc $ docker push urik/web-uoc
The push refers to a repository [urik/web-uoc] (len: 1)
d7e1c2ffd4b7: Image already exists
6371d1f8981b: Image successfully pushed
5e8c2d826b70: Image successfully pushed
44df5ce8083a: Image successfully pushed
de401b043452: Image successfully pushed
6705ebcea7a3: Image successfully pushed
d80ddc7f43a1: Image successfully pushed
6796b7682b3e: Image already exists
b30cb5f1d088: Image already exists
bae2b1c607c7: Image already exists
d74c94329c2b: Image already exists
181e8ba79abb: Image successfully pushed
c8aab46f49f7: Image successfully pushed
1db7de304924: Image successfully pushed
0f062bc85662: Image successfully pushed
ea6bab360f56: Image already exists
5679b9b90e09: Image successfully pushed
Digest: sha256:7b04fbe5c83ddf0b91b2d2d445c0fb2a0a94d15dc6ac6690bbae822d7ad03b3a
```

Ara ja podem executar un contenidor amb aquesta imatge des de qualsevol servidor. Per fer-ho executem:

```
docker run -d -p 8080 -m 200M --env VHOST=$VHOST urik/web-uoc
```

Si encara no tenim la imatge en local la descarregarà del repositori i si ja la tenim arrencarà directament. Aquest és el motiu pel qual en els scripts de control, quan arrenquem per primera vegada un contenidor, el script tarda uns segons ja que està descarregant la imatge en local. En canvi quan arranquem el segon contenidor sempre va molt més ràpid. També podem baixar una imatge abans d'executar-la amb la següent comanda.

```
docker pull urik/web-uoc
```

En aquest cas com que ja la tenim creada en local no es baixa res.

```
core@control ~/autoscalableswarm/web-uoc $ docker pull urik/web-uoc
latest: Pulling from urik/web-uoc
5679b9b90e09: Already exists
ea6bab360f56: Already exists
```

```
0f062bc85662: Already exists
1db7de304924: Already exists
c8aab46f49f7: Already exists
181e8ba79abb: Already exists
d74c94329c2b: Already exists
bae2b1c607c7: Already exists
b30cb5f1d088: Already exists
6796b7682b3e: Already exists
d80ddc7f43a1: Already exists
6705ebcea7a3: Already exists
de401b043452: Already exists
44df5ce8083a: Already exists
5e8c2d826b70: Already exists
6371d1f8981b: Already exists
d7e1c2ffd4b7: Already exists
Digest: sha256:7b04fbe5c83ddf0b91b2d2d445c0fb2a0a94d15dc6ac6690bbae822d7ad03b3a
Status: Image is up to date for urik/web-uoc:latest
```

Tot el codi del contenidor Web UOC el podem trobar en el repositori de git https://bitbucket.org/urik23/autoscalableswarm.git dins de la carpeta web-uoc.

Swarm Master

Una vegada tenim definida i creada la base del sistema (Servidor de Control i NGINX) ja podem inicialitzar el clúster de Swarm. Per fer-ho necessitem instanciar un node que farà de master del clúster i que ens permetrà crear i destruir tots els nodes que necessitem. El servidor swarm-master s'executa sobre un ubuntu on tindrem instal·lat docker i executarem els contenidors. De moment només es pot crear sobre ubuntu però ja està pensat que en properes versions es pugui executar sobre CoreOS. Per defecte el swarm master arrenca amb dos contenidors per gestionar el clúster. I després arrencarem els dos contenidors amb la nostra aplicació web.

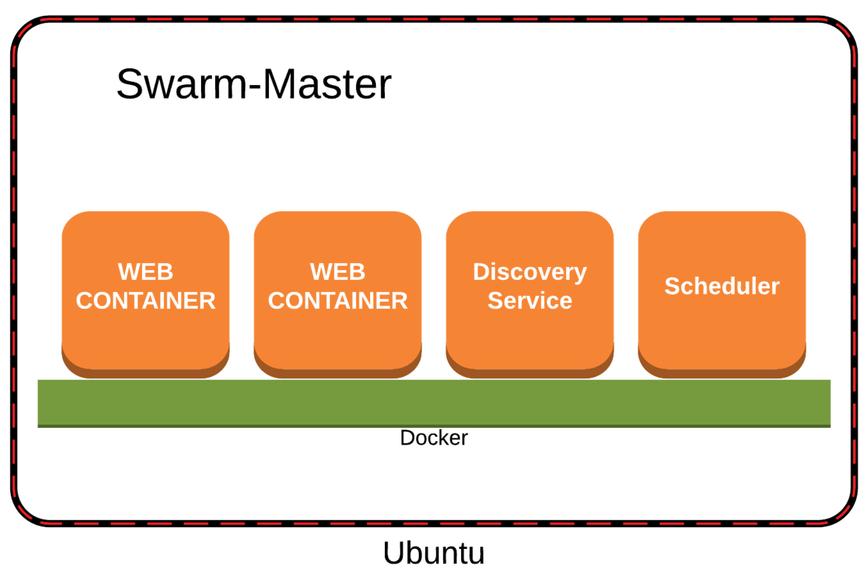

Figura 11 – Swarm Master

Per crear el swarm master des del servidor de control hem d'executar la següent comanda.

```
docker-machine create -d digitalocean --digitalocean-access-token
6a96d93550c7f6dd4e89472b2f0974d34687cf7ea7489d95cd41d42720a03967 --swarm --swarm-master --swarm-discovery
token://dcea60ab8f5fffd12559c981900a3ae6 swarm-master
```

-d digitalocean: indiquem el driver per connectar amb el cloud

--digitalocean-access-token: Li passem el token que hem generat a la web de digital ocean per poder-nos verificar i connectar

--swarm: Per indicar que volem instal·lar swarm en el servidor que creem i que aquest sigui el master.

--*swarm-discovery*: És un token generat per identificar el clúster de swarm que estem creant.
I l'últim paràmetre és el nom de la màquina.

Swarm Node

Els nodes de swarm també es creen des del servidor de control i s'executen sobre un ubuntu. En aquest cas només tenim un contenidor per la gestió i després tenim els dos contenidors que executen l'aplicació web. Es pot donar el cas que només en tinguem un corrent però com a màxim en podem tenir dos.

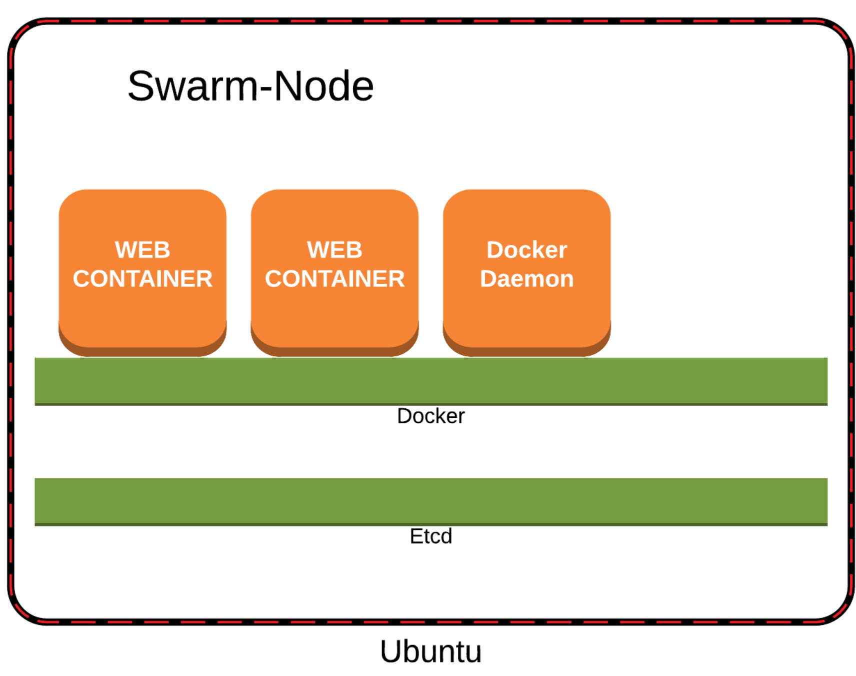

Figura 12 – Swarm Node

Proves de Rendiment

Per poder fer les proves de rendiment s’ha publicat una documentació detallada de com instal·lar i testejar tot el sistema en un repositori públic de git.

Es pot consultar a: https://urik23@bitbucket.org/urik23/autoscalableswarm.git

Per descarregar el repositori només hem d’executar des d’un pc que tingui git instal·lat.

```
git clone https://urik23@bitbucket.org/urik23/autoscalableswarm.git
```

Per fer les proves farem servir jMeter que ens permet llençar peticions de forma continuada simulant diferents usuaris.

El fitxer de configuració de jMeter que utilitzarem i que està disponible en el repositori és el següent.


```
<?xml version="1.0" encoding="UTF-8"?>
<jmeterTestPlan version="1.2" properties="2.8" jmeter="2.13 r1665067">
  <hashTree>
    <TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="Test Plan" enabled="true">
      <stringProp name="TestPlan.comments"></stringProp>
      <boolProp name="TestPlan.functional_mode">false</boolProp>
      <boolProp name="TestPlan.serialize_threadgroups">false</boolProp>
      <elementProp name="TestPlan.user_defined_variables" elementType="Arguments" guiclass="ArgumentsPanel"
testclass="Arguments" testname="User Defined Variables" enabled="true">
        <collectionProp name="Arguments.arguments"/>
      </elementProp>
      <stringProp name="TestPlan.user_define_classpath"></stringProp>
    </TestPlan>
    <hashTree>
      <ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup" testname="Thread Group" enabled="true">
        <stringProp name="ThreadGroup.on_sample_error">continue</stringProp>
        <elementProp name="ThreadGroup.main_controller" elementType="LoopController" guiclass="LoopControlPanel"
testclass="LoopController" testname="Loop Controller" enabled="true">
          <boolProp name="LoopController.continue_forever">false</boolProp>
          <intProp name="LoopController.loops">-1</intProp>
        </elementProp>
        <stringProp name="ThreadGroup.num_threads">4</stringProp>
        <stringProp name="ThreadGroup.ramp_time">1</stringProp>
        <longProp name="ThreadGroup.start_time">1446825126000</longProp>
        <longProp name="ThreadGroup.end_time">1446825126000</longProp>
        <boolProp name="ThreadGroup.scheduler">false</boolProp>
        <stringProp name="ThreadGroup.duration"></stringProp>
        <stringProp name="ThreadGroup.delay"></stringProp>
      </ThreadGroup>
      <hashTree>
        <ConfigTestElement guiclass="HttpDefaultsGui" testclass="ConfigTestElement" testname="HTTP Request Defaults"
enabled="true">
          <elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel"
```

```
testclass="Arguments" testname="User Defined Variables" enabled="true">
            <collectionProp name="Arguments.arguments"/>
          </elementProp>
          <stringProp name="HTTPSampler.domain">46.101.75.148</stringProp>
          <stringProp name="HTTPSampler.port"></stringProp>
          <stringProp name="HTTPSampler.connect_timeout"></stringProp>
          <stringProp name="HTTPSampler.response_timeout"></stringProp>
          <stringProp name="HTTPSampler.protocol"></stringProp>
          <stringProp name="HTTPSampler.contentEncoding"></stringProp>
          <stringProp name="HTTPSampler.path"></stringProp>
          <stringProp name="HTTPSampler.concurrentPool">4</stringProp>
        </ConfigTestElement>
        <hashTree/>
        <HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy" testname="HTTP Request"
enabled="true">
          <elementProp name="HTTPsampler.Arguments" elementType="Arguments" guiclass="HTTPArgumentsPanel"
testclass="Arguments" testname="User Defined Variables" enabled="true">
            <collectionProp name="Arguments.arguments"/>
          </elementProp>
          <stringProp name="HTTPSampler.domain"></stringProp>
          <stringProp name="HTTPSampler.port"></stringProp>
          <stringProp name="HTTPSampler.connect_timeout"></stringProp>
          <stringProp name="HTTPSampler.response_timeout"></stringProp>
          <stringProp name="HTTPSampler.protocol"></stringProp>
          <stringProp name="HTTPSampler.contentEncoding"></stringProp>
          <stringProp name="HTTPSampler.path">/fibonacci/35</stringProp>
          <stringProp name="HTTPSampler.method">GET</stringProp>
          <boolProp name="HTTPSampler.follow_redirects">true</boolProp>
          <boolProp name="HTTPSampler.auto_redirects">false</boolProp>
          <boolProp name="HTTPSampler.use_keepalive">true</boolProp>
          <boolProp name="HTTPSampler.DO_MULTIPART_POST">false</boolProp>
          <boolProp name="HTTPSampler.monitor">false</boolProp>
          <stringProp name="HTTPSampler.embedded_url_re"></stringProp>
        </HTTPSamplerProxy>
        <hashTree/>
        <ResultCollector guiclass="TableVisualizer" testclass="ResultCollector" testname="View Results in Table"
enabled="true">
          <boolProp name="ResultCollector.error_logging">false</boolProp>
          <objProp>
            <name>saveConfig</name>
            <value class="SampleSaveConfiguration">
              <time>true</time>
              <latency>true</latency>
              <timestamp>true</timestamp>
              <success>true</success>
              <label>true</label>
              <code>true</code>
              <message>true</message>
              <threadName>true</threadName>
              <dataType>true</dataType>
              <encoding>false</encoding>
              <assertions>true</assertions>
```

```
            <subresults>true</subresults>
            <responseData>false</responseData>
            <samplerData>false</samplerData>
            <xml>false</xml>
            <fieldNames>false</fieldNames>
            <responseHeaders>false</responseHeaders>
            <requestHeaders>false</requestHeaders>
            <responseDataOnError>false</responseDataOnError>
            <saveAssertionResultsFailureMessage>false</saveAssertionResultsFailureMessage>
            <assertionsResultsToSave>0</assertionsResultsToSave>
            <bytes>true</bytes>
            <threadCounts>true</threadCounts>
          </value>
        </objProp>
        <stringProp name="filename"></stringProp>
      </ResultCollector>
      <hashTree/>
    </hashTree>
  </hashTree>
 </hashTree>
</jmeterTestPlan>
```

El primer que fem és arrencar el sistema amb els dos contenidors base. Un cop fets tots els passos del document d'instal·lació tenim el següent estat on podem veure el node swarm master i els dos contenidors corrents amb id *c5528ef6822d* i *4d82cf2a04e6*

```
core@control ~ $ docker-machine ls
NAME           ACTIVE   DRIVER         STATE     URL                          SWARM
swarm-master   -        digitalocean   Running   tcp://178.62.62.143:2376     swarm-master (master)
core@control ~ $ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED              STATUS               PORTS
NAMES
c5528ef6822d            urik/web-uoc            "java -Djava.securit     About a minute ago     Up About a minute
178.62.62.143:32769->8080/tcp   swarm-master/pensive_kowalevski
4d82cf2a04e6            urik/web-uoc            "java -Djava.securit     About a minute ago     Up About a minute
178.62.62.143:32768->8080/tcp   swarm-master/boring_jennings
```

En la pàgina de gestió de digital ocean podem veure també els servidors.

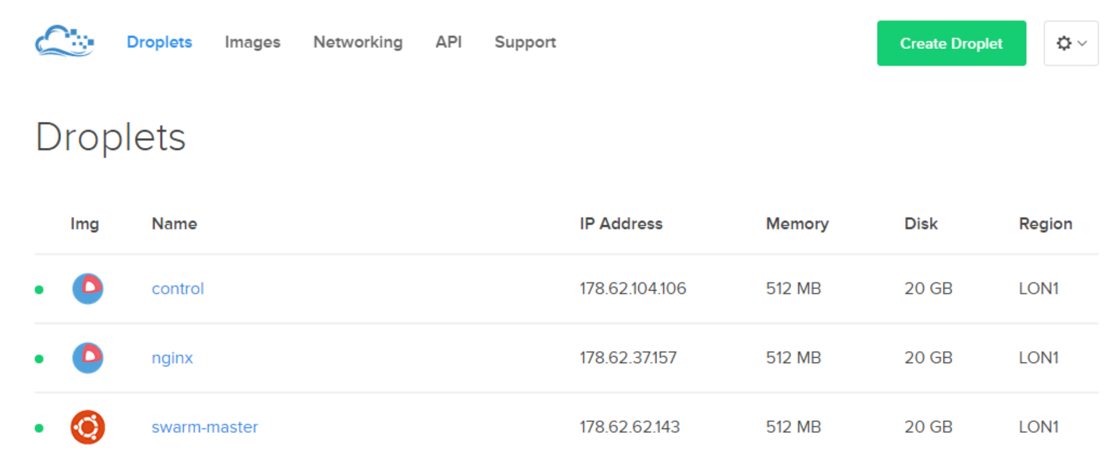

Figura 13 – Digital Ocean Web

Per comprovar que tot està correcte accedim per web a la ip del NGINX i mirem si ens està contestant el contenidor *4d82cf2a04e6*.

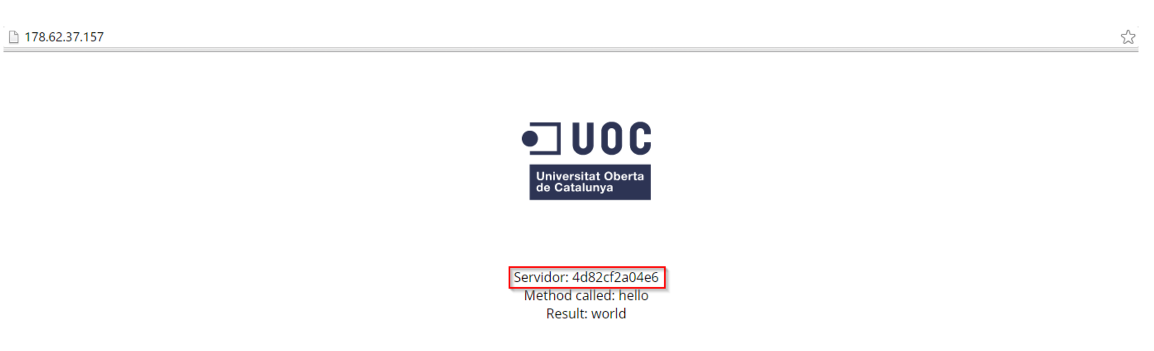

Figura 14 – Web Contenidor

Si tornem a accedir ens retorna la pàgina l'altre contenidor *c5528ef6822d*.

178.62.37.157

Servidor: c5528ef6822d
Method called: hello
Result: world

Figura 15 – Web Contenidor

Els llindars que tenim definits per la CPU són mínim 1 màxim 2. Per tant si la mitja està per sota de 1 hauríem d'eliminar servidors i si està per sobre de 2 crear-ne. Per comprovar-ho executem la següent comanda.

```
core@control ~ $ etcdctl get /GLOBAL/MIN_CPU_CORE
1
core@control ~ $ etcdctl get /GLOBAL/MAX_CPU_CORE
2
```

Un cop comprovat que tot està correcte ja podem executar el script per comprovar la CPU.

```
core@control ~ $ checkCpu.sh
0.0725
var1  0
var2  1
Comprobem quants servidors tenim:  1
```

Com que ningú està accedint la mitja és de 0.0725. Està per sota de 1 però com que només tenim el Swarm Master no podem eliminar cap servidor.

Amb l'ajut del jMeter començarem a llençar peticions contra el clúster per mirar de fer augmentar la CPU i comprovar que això provoca que es creï un nou servidor. Llençarem 4 peticions cada segon.

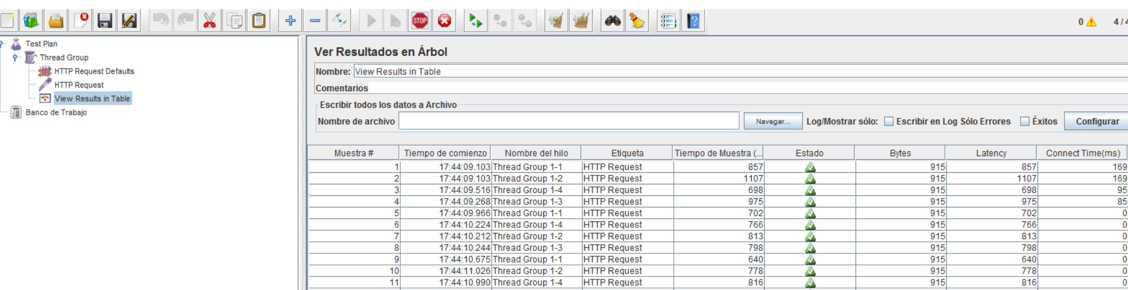

Figura 16 – Pantalla Jmeter

Tornem a llençar el checkCPU per comprovar en quin estat estem.

```
core@control ~ $ checkCpu.sh
1.06833
var1  0
var2  0
core@control ~ $ checkCpu.sh
1.5825
var1  0
var2  0
core@control ~ $ checkCpu.sh
1.904
var1  0
var2  0
core@control ~ $ checkCpu.sh
2.578
var1  1
StartContainer
var2  0
```

Podem comprovar que la CPU en cada execució va pujant fins sobrepassar l'umbral màxim, cosa que provoca intentar arrencar un nou contenidor. Com que no ho podem fer en la infraestructura actual ens obliga a crear un nou servidor per poder executar el nou contenidor.

Això ho podem comprovar amb les següents comandes.

Primer comprovem que s'ha creat un nou servidor.

```
core@control ~ $ docker-machine ls
NAME           ACTIVE   DRIVER         STATE     URL                         SWARM
swarm-master   -        digitalocean   Running   tcp://178.62.62.143:2376    swarm-master (master)
swarm-node-1   -        digitalocean   Running   tcp://46.101.35.55:2376     swarm-master
```

Comprovem que estigui disponible en el clúster de swarm.

```
core@control ~ $ docker info
Containers: 6
Images: 4
Role: primary
Strategy: spread
Filters: health, port, dependency, affinity, constraint
Nodes: 2
 swarm-master: 178.62.62.143:2376
  └ Containers: 4
  └ Reserved CPUs: 0 / 1
  └ Reserved Memory: 400 MiB / 514.5 MiB
  └ Labels: executiondriver=native-0.2, kernelversion=3.13.0-68-generic, operatingsystem=Ubuntu 14.04.3 LTS,
provider=digitalocean, storagedriver=aufs
 swarm-node-1: 46.101.35.55:2376
  └ Containers: 2
  └ Reserved CPUs: 0 / 1
```

```
  └ Reserved Memory: 200 MiB / 514.5 MiB
  └  Labels:  executiondriver=native-0.2,  kernelversion=3.13.0-68-generic,  operatingsystem=Ubuntu  14.04.3  LTS,
provider=digitalocean, storagedriver=aufs
CPUs: 2
Total Memory: 1.005 GiB
Name: bc35e293342c
```

Comprovem que el contenidor s'està executant i que ja en tenim tres disponibles.

```
core@control ~ $ docker ps
CONTAINER ID          IMAGE                    COMMAND                    CREATED                STATUS                 PORTS
NAMES
b9cbdde2f1fe              urik/web-uoc              "java  -Djava.securit     2  minutes  ago             Up  2  minutes
46.101.35.55:32768->8080/tcp     swarm-node-1/jovial_swanson
c5528ef6822d              urik/web-uoc              "java  -Djava.securit     30  minutes  ago           Up  30  minutes
178.62.62.143:32769->8080/tcp    swarm-master/pensive_kowalevski
4d82cf2a04e6              urik/web-uoc              "java  -Djava.securit     30  minutes  ago           Up  30  minutes
178.62.62.143:32768->8080/tcp    swarm-master/boring_jennings
```

Tornem a executar el checkCPU per veure l'estat. Com que encara és alta es crea un altre contenidor.

```
core@control ~ $ checkCpu.sh
2.969
var1  1
StartContainer
var2  0
core@control ~ $ docker ps
CONTAINER ID          IMAGE                    COMMAND                    CREATED                STATUS                 PORTS
NAMES
eac89def761a              urik/web-uoc              "java  -Djava.securit     13  seconds  ago           Up  13  seconds
46.101.35.55:32769->8080/tcp     swarm-node-1/clever_morse
b9cbdde2f1fe              urik/web-uoc              "java  -Djava.securit     7  minutes  ago             Up  7  minutes
46.101.35.55:32768->8080/tcp     swarm-node-1/jovial_swanson
c5528ef6822d              urik/web-uoc              "java  -Djava.securit     34  minutes  ago           Up  34  minutes
178.62.62.143:32769->8080/tcp    swarm-master/pensive_kowalevski
4d82cf2a04e6              urik/web-uoc              "java  -Djava.securit     34  minutes  ago           Up  34  minutes
178.62.62.143:32768->8080/tcp    swarm-master/boring_jennings
```

Si comprovem els servidor que tenim actualment a Digital Ocean veurem que en tenim un de nou on s'estan executant dos contenidors més.

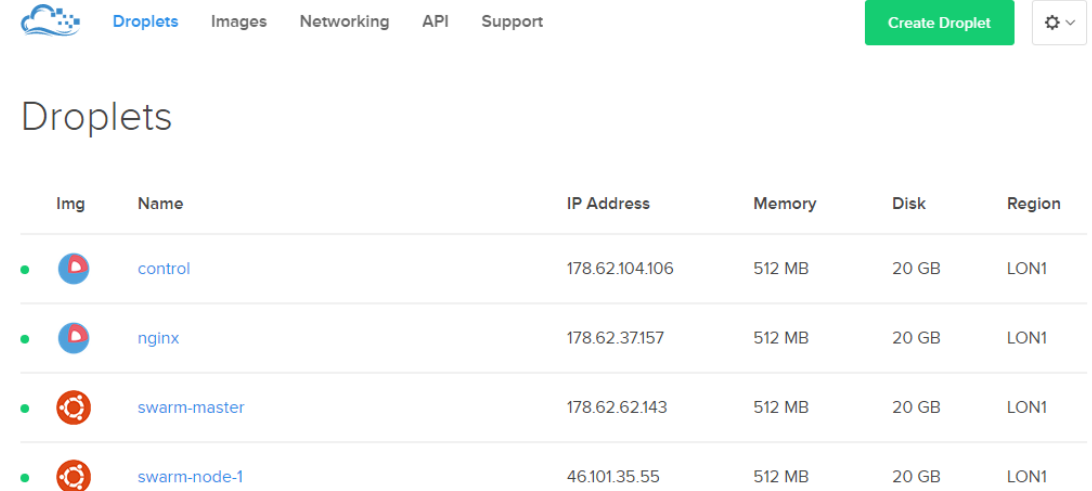

Figura 17 – Digital Ocean Web

Ara parem el jMeter i deixem d'enviar peticions al clúster i comprovem l'estat de la CPU.

```
core@control /opt/bin $ checkCpu.sh
1.4552
var1  0
var2  0
core@control /opt/bin $ checkCpu.sh
0.127067
var1  0
var2  1
Comprobem quants servidors tenim:  2
Destruïm Servidor
core@control /opt/bin $
```

Com que ja no tenim els accessos als servidors, la CPU baixa fins al punt de quedar per sota del mínim i això provoca que eliminem el servidor. Per tant tornem a quedar-nos només amb els dos contenidors inicials.

```
core@control /opt/bin $ docker-machine ls
NAME          ACTIVE   DRIVER         STATE     URL                         SWARM
swarm-master  -        digitalocean   Running   tcp://178.62.62.143:2376    swarm-master (master)
core@control /opt/bin $ docker info
Containers: 4
Images: 2
Role: primary
Strategy: spread
Filters: health, port, dependency, affinity, constraint
Nodes: 1
 swarm-master: 178.62.62.143:2376
  └ Containers: 4
  └ Reserved CPUs: 0 / 1
```

```
  └ Reserved Memory: 400 MiB / 514.5 MiB
  └  Labels:  executiondriver=native-0.2,  kernelversion=3.13.0-68-generic,  operatingsystem=Ubuntu  14.04.3  LTS,
provider=digitalocean, storagedriver=aufs
CPUs: 1
Total Memory: 514.5 MiB
Name: bc35e293342c
core@control /opt/bin $ docker ps
CONTAINER ID         IMAGE                 COMMAND                   CREATED              STATUS               PORTS
NAMES
c5528ef6822d             urik/web-uoc            "java  -Djava.securit     47  minutes  ago          Up  47  minutes
178.62.62.143:32769->8080/tcp   swarm-master/pensive_kowalevski
4d82cf2a04e6             urik/web-uoc            "java  -Djava.securit     47  minutes  ago          Up  47  minutes
178.62.62.143:32768->8080/tcp   swarm-master/boring_jennings
```

I comprovem que en l'entorn de Digital Ocean també s'ha eliminat i tornem a estar com al principi.

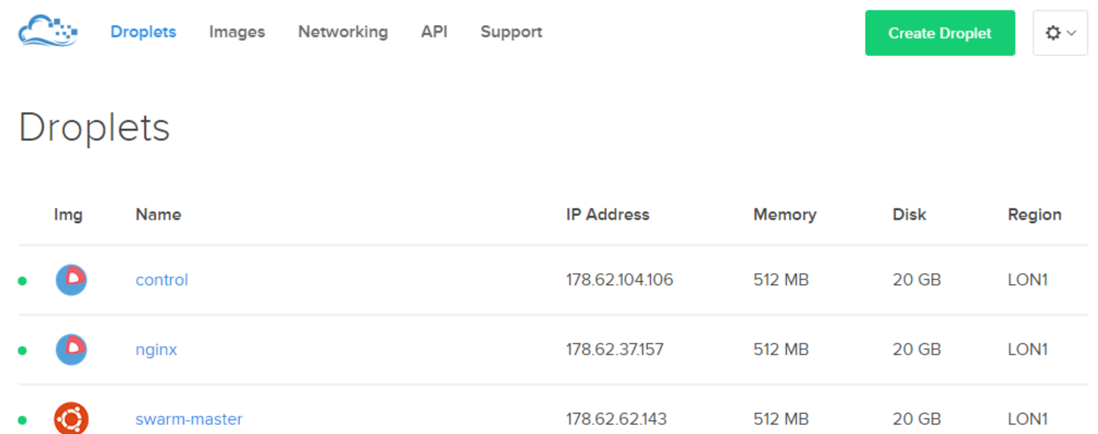

Figura 18 – Digital Ocean Web

Millores

La realització d'aquest projecte ens ha permès comprovar que Docker i Swarm ens facilitarien la feina a l'hora de crear i administrar un clúster autoescalable tot i que el disseny utilitzat no es podria traslladar directament a un entorn productiu.

S'haurien de realitzar alguns canvis que fessin el sistema més redundant ja que com està pensat actualment, encara que per un estudi acadèmic és vàlid, no dóna suficients garanties en un entorn productiu. Com es pot veure en el disseny original hi ha alguns punts crítics que en cas de fallada ens podrien fer caure tot el clúster. Aquests bàsicament són tres; el servidor NGINX, el servidor de Control i el Swarm Master.

Per mirar de solucionar aquests problemes es proposa el següent disseny.

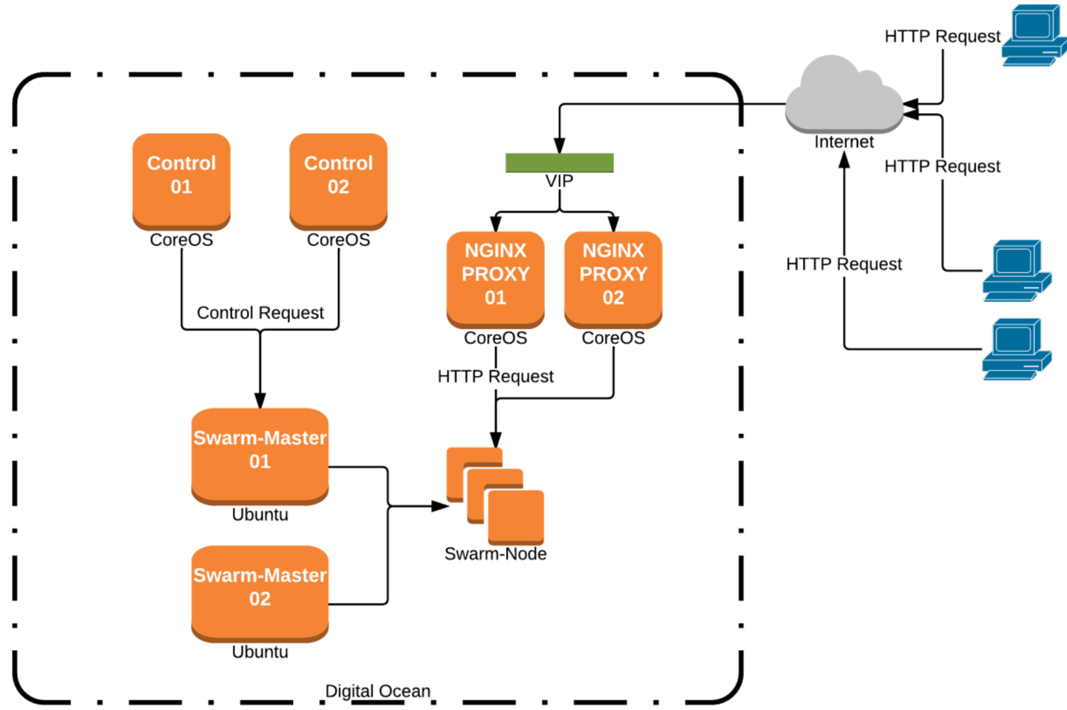

Figura 19 – Arquitectura Millorada

Com podem comprovar la base és la mateixa però s'han duplicat servidors. El primer punt és el proxy NGINX. Aquí proposem fer servir una IP virtual que apuntarà a un dels dos nodes de NGINX. Si per algun motiu el principal falla la IP virtual migrarà de node i seguirem donant servei pel secundari. El disseny que es proposa és un model actiu passiu i que només en cas de caiguda s'activi el segon node. Per resoldre aquest punt Digital Ocean ha creat el concepte de Floating IP que permet assignar una IP a un droplet com a principal i definir un secundari. La IP

apunta al principal i en cas de caiguda la migra automàticament al secundari. En cas de no utilitzar Digital Ocean hi ha múltiples opcions per realitzar el mateix com per exemple keepalived.

El segon punt de millora és el servidor de control. En aquest cas com que hem utilitzat etcd per definir les variables d'entorn ens facilitarà molt la feina ja que els dos servidors podran executar els scripts que hem preparat. Per exemple, la variable que utilitzem per saber el nombre de nodes swarm que tenim és visible i actulitzable pels dos nodes a través de etcd per tant podríem arrencar un nou servidor des del control 01 i parar-lo des del control 02 sense cap problema. Per no haver de tenir els scripts duplicats es podria crear un contenidor que seria l'encarregat d'executar aquests scripts i seria el contenidor que mouríem entre els nodes si un fallés. En aquest cas també es definiria un entorn actiu passiu on només en cas de caiguda migraríem el contenidor al segon node.

Un altre punt crític és el Swarm Master ja que sense aquest servidor perdem el control de tots els nodes de Swarm. Per aquest motiu Swarm ja contempla una configuració amb múltiples Swarm Master on un d'ells té el control però sempre el pot delegar a un dels altres. Tenim un Swarm Master primari i després N rèpliques que són candidats a fer de swarm master. Aquests nodes, com el master, també poden executar contenidors per tant els podem aprofitar per la nostra aplicació i a sobre en cas de caiguda del primari aquest cedirà el control a un dels altres.

I per últim un altre punt de millora seria el sistema com valorem l'estat del clúster. En el nostre cas hem escollit la CPU però segur que hi ha més factors que defineixen si un entorn està saturat o no. Per tant, en cada cas, s'hauria d'analitzar bé com i quan decidim que no donem un servei correcte i llavors executar l'ampliació del nostre clúster.

Durant la realització d'aquest projecte la tecnologia Swarm estava en fase beta però en el moment d'escriure aquest document ha passat a producció. Amb l'última actualització de docker, la 1.9.0, s'ha afegit Swarm per tant creiem que això serà també una millora ja que segur que serà més estable i probablement s'hauran afegit més funcionalitats que les utilitzades en aquest projecte.

Rendiment

Un cop realitzat el projecte hem d'analitzar si ens aporta els beneficis que havíem previst. Per fer-ho agafarem com a base una empresa de turisme que té el gruix important del seu negoci en la venda online d'habitacions d'hotel. Aquesta empresa té uns mesos de temporada molt alta i la resta de l'any baixa molt el tràfic que ha de suportar. També té grans oscil·lacions entre les hores diürnes i nocturnes. Aquest fet la fa una empresa ideal per implementar el nostre sistema ja que a la temporada baixa estalviarà recursos i en temporada alta si en algun moment té una pujada més forta de la planificada ho podrà suportar i seguir venent habitacions.

Suposem que les temporades per mesos són les següents:

Mes	Temporada
Gener	Baixa
Febrer	Baixa
Març	Alta
Abril	Baixa
Maig	Alta
Juny	Alta
Juliol	Alta
Agost	Alta
Setembre	Baixa
Octubre	Baixa
Novembre	Baixa
Desembre	Alta

Taula 7 - Temporada Mesos

I suposem que la franja horària de més tràfic és de 12:00h a 24:00h. Per tant hem de dimensionar el nostre sistema perquè pugui suportar la càrrega d'un mes d'agost a les 22h. Suposem que per suportar aquesta càrrega necessitem 8 servidors de 16 Gb de memòria i 8 processadors.

Aquests servidors a Digital Ocean tenen un cost de 160€ al mes. Per tant l'empresa anualment haurà de pagar 15.360€.

Servidors	Cost Unitari	Cost Mensual	Cost Anual
8	160€	1280€	15.360€

Taula 8 - Cost Servidors Sense Clúster

Si aquesta empresa implementa el nostre sistema el cost total dels servidors hauria de disminuir.

Ells han calculat que en temporada baixa amb la meitat de la infraestructura ja poden donar un bon servei. I que dins la mateixa temporada a les hores baixes poden reduir un 25% de la seva infraestructura sense baixar el rendiment.

Si tenim en compte aquests paràmetres en un mes de temporada baixa necessiten 4 servidors la meitat del temps i l'altra meitat amb 3 servidors ja és suficient. Per tant, podem deduir que necessiten una mitjana de 3`5 servidors el mes de temporada baixa.

El cost mensual és de 560€

Servidors	Cost unitari	Cost Mensual
3,5	160€	560€

Taula 9 - Mes temporada baixa

Per un mes de temporada alta necessiten 8 servidors la meitat del mes i l'altra meitat amb 6 servidors ja és suficient. Per tant surt una mitja de 7 servidors en un mes de temporada alta.

El cost mensual és de 1.120€

Servidors	Cost unitari	Cost Mensual
7	160€	1.120€

Taula 10 - Mes temporada alta

Un cop calculats els costos de cada mes podem saber el total anual que haurien de pagar.

Mes	Temporada	Cost Mensual
Gener	Baixa	560€
Febrer	Baixa	560€
Març	Alta	1.120€
Abril	Baixa	560€
Maig	Alta	1.120€
Juny	Alta	1.120€
Juliol	Alta	1.120€
Agost	Alta	1.120€
Setembre	Baixa	560€
Octubre	Baixa	560€
Novembre	Baixa	560€
Desembre	Alta	1.120€
	Total Anual:	**10.080€**

El cost anual de la infraestructura amb el clúster autoescalable surt per 10.080€ anuals. Per tant, ens estalviem 5.280€ l'any, un 34% d'estalvi directe en la infraestructura de servidors sense perdre capacitat de negoci.

I amb l'avantatge que si en algun moment tenim necessitat d'acceptar més tràfic, ja que s'ha fet alguna campanya d'ofertes, es podrà ampliar encara més el sistema i podrem resoldre totes les peticions sense la necessitat de descartar-ne cap. Amb el sistema tradicional això no seria possible ja que no podriem reaccionar a temps.

	Sense Clúster	**Amb Clúster**	
Cost Anual	15.360€	10.080€	Un 34% d'estalvi
Flexibilitat	Baixa	Alta	Adaptació a la carrega dinamicament

Taula 11 - Millores Clúster

Queda clar que amb el clúster autoescalable reduïm els costos de forma significativa i tenim més marge de reacció per si la demanada puja en qualsevol moment. Aquests dos punts son molt importants per fer viable aquest projecte a la direcció de qualsevol empresa.

Valoració

La conclusió principal que podem extreure després de realitzar aquest projecte és que amb les eines que tenim actualment es pot dissenyar i implementar un clúster autoescalable de forma senzilla. Aquest mateix plantejament fa més de dos anys hagués estat molt més complicat. Això és degut a diferents qüestions. La primera és l'aparició d'un sistema de virtualització de programari. Fins ara teníem la virtualització de maquinari, com VMWare, però s'ha fet un pas més i el que s'ha aconseguit és empaquetar en un contenidor tot el necessari per poder executar una aplicació.

Això té moltes avantatges ja que un cop has garantit el funcionament de l'aplicació la pots executar a diferents entorns i saps que el comportament sempre serà el mateix. També permet tenir un control de versions millorat ja que el que versiones és l'aplicació més tots els programes, plugins, drivers, etc... que necessites per executar-la. Si a l'executar una nova versió tens algun problema només cal aturar-la i executar l'anterior sense haver de reinstal·lar res en el servidor.

També aporta avantatges en el Continous Delivery ja que és relativament fàcil, a través de scripts o d'eines com Jenkins, fer processos que s'executin cada cop que es fa commit d'una nova versió de codi. D'aquesta manera estem provant contínuament que tot el que s'està fent es pot executar correctament en el nostre entorn. Segur que hi ha molts més motius però aquests són suficients per dir que l'ús de contenidors i en concret de Docker revolucionarà la manera de gestionar el manteniment de les aplicacions.

En el nostre cas gràcies a Docker podem desplegar de forma senzilla i ràpida la nostra aplicació cada cop que ho necessitem.Un altre punt important és l'aparició dels datacenter en el núvol. El fet de no haver de gestionar les màquines en un CPD tradicional ens permet una gran flexibilitat i una reducció de costos. Flexibilitat perquè si necessitem més servidors només s'han de crear. Abans els passos eren molt més lents i en molts casos comportaven un cost més elevat.I sumat a tot això tenim dos eines clau que són Swarm i Machine. Per una banda Machine ens simplifica el fet d'aprovisionar nous servidors en el cloud i poder-los gestionar d'una forma conjunta. I swarm ens permet tractar una granja de servidors Docker com si només en tinguéssim un.

Aquests dos fets simplifiquen molt la gestió del sistema ja que ens permeten crear nous servidors de forma ràpida i gestionar conjuntament tots els contenidors que hi executem. Ara que aquestes dos eines comencen a estar disponibles per entorns productius segur que es trobaran moltes més utilitats que la que proposa aquest projecte. Només cal veure l'evolució que ha tingut en dos anys tota aquesta nova manera de veure la gestió del programari i la gran implementació que està tenint en tot tipus d'empresa, des de les més petites a les més grans, per saber que té molt més recorregut. Després de passar moltes hores llegint, investigant i

provant m’ha quedat clar que estem davant d’una nova tecnologia que de ben segur serà el futur, com en el seu dia va ser la virtualització.

I també hem pogut demostrar que l'ús d'aquesta arquitectura aporta un benefici econòmic per l'empresa. Això és degut a que podem reduir l'ús dels servidors en temporades baixes on no és necessari tenir tota la infraestructura operativa. Aquest fet provoca una reducció del 34% del cost dels servidors utilitzats. A part permet l'ampliació de la infraestructura en moments de màxima demanda poden donar servei a tots els usuaris i augmentant així la possibilitat de obtenir més ingressos.

Per tant hem aconseguit un sistema fàcil de configurar i gestionar i que també aporta un benifici directe a l'empresa. Hem complert dos dels objectius que es van plantejar al inici d'aquest projecte.

Annex A

Sortida completa de la comanda:

```
docker build -t urik/nginx-proxy-swarm .
```

```
core@control ~/autoscalableswarm/nginx-proxy-swarm $ docker build -t urik/nginx-proxy-swarm .
Sending build context to Docker daemon 10.24 kB
Sending build context to Docker daemon
Step 0 : FROM nginx:1.9.5
1.9.5: Pulling from nginx
575489a51992: Pull complete
6845b83c79fb: Pull complete
d0ea6b0b9e6a: Pull complete
77f88505fa0d: Pull complete
98de8af78960: Pull complete
fadf699bccc0: Pull complete
5ac100425925: Pull complete
738e7a471e87: Pull complete
e240c7325698: Pull complete
5cd1c00ad84f: Pull complete
c1ff64fa1aa6: Pull complete
3669a1f15a6f: Pull complete
nginx:1.9.5: The image you are pulling has been verified. Important: image verification is a tech preview feature and
should not be relied on to provide security.
Digest: sha256:991bdd670fb03e0133cd72fd17add6622803eb904339d6ae9076aee402d71519
Status: Downloaded newer image for nginx:1.9.5
 ---> 3669a1f15a6f
Step 1 : MAINTAINER Oriol Moya Ahufinger oriol.moya@gmail.com
 ---> Running in cb9585638633
 ---> 6ca38d803cb4
Removing intermediate container cb9585638633
Step 2 : RUN mkdir -p /opt/bin
 ---> Running in dc870978b65c
 ---> 207a81d47bd5
Removing intermediate container dc870978b65c
Step 3 : RUN mkdir -p /certs
 ---> Running in 81cd07f5db72
 ---> 42116abf6dd6
Removing intermediate container 81cd07f5db72
Step 4 : RUN mkdir -p /template
 ---> Running in 2afe62e953df
 ---> 3ec74dd1dea7
Removing intermediate container 2afe62e953df
Step 5 : RUN apt-get update  && apt-get install -y -q --no-install-recommends     wget
 ---> Running in 04ab2d00fd1c
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://httpredir.debian.org jessie-updates InRelease [135 kB]
Get:3 http://httpredir.debian.org jessie Release.gpg [2373 B]
Get:4 http://httpredir.debian.org jessie Release [148 kB]
Get:5 http://nginx.org jessie Release.gpg [287 B]
Get:6 http://nginx.org jessie Release [2313 B]
Get:7 http://security.debian.org jessie/updates/main amd64 Packages [189 kB]
Get:8 http://httpredir.debian.org jessie-updates/main amd64 Packages [3619 B]
Get:9 http://httpredir.debian.org jessie/main amd64 Packages [9035 kB]
Get:10 http://nginx.org jessie/nginx amd64 Packages [1109 B]
Fetched 9581 kB in 4s (2287 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
The following extra packages will be installed:
  libffi6 libgmp10 libgnutls-deb0-28 libhogweed2 libicu52 libidn11 libnettle4
  libp11-kit0 libpsl0 libtasn1-6
Suggested packages:
  gnutls-bin
The following NEW packages will be installed:
  libffi6 libgmp10 libgnutls-deb0-28 libhogweed2 libicu52 libidn11 libnettle4
  libp11-kit0 libpsl0 libtasn1-6 wget
0 upgraded, 11 newly installed, 0 to remove and 1 not upgraded.
Need to get 8856 kB of archives.
After this operation, 34.2 MB of additional disk space will be used.
Get:1 http://security.debian.org/ jessie/updates/main libicu52 amd64 52.1-8+deb8u3 [6784 kB]
Get:2 http://httpredir.debian.org/debian/ jessie/main libgmp10 amd64 2:6.0.0+dfsg-6 [253 kB]
Get:3 http://httpredir.debian.org/debian/ jessie/main libnettle4 amd64 2.7.1-5 [176 kB]
Get:4 http://httpredir.debian.org/debian/ jessie/main libhogweed2 amd64 2.7.1-5 [125 kB]
Get:5 http://httpredir.debian.org/debian/ jessie/main libffi6 amd64 3.1-2+b2 [20.1 kB]
Get:6 http://httpredir.debian.org/debian/ jessie/main libp11-kit0 amd64 0.20.7-1 [81.2 kB]
Get:7 http://httpredir.debian.org/debian/ jessie/main libtasn1-6 amd64 4.2-3+deb8u1 [48.9 kB]
Get:8 http://httpredir.debian.org/debian/ jessie/main libgnutls-deb0-28 amd64 3.3.8-6+deb8u3 [694 kB]
Get:9 http://httpredir.debian.org/debian/ jessie/main libidn11 amd64 1.29-1+b2 [136 kB]
Get:10 http://httpredir.debian.org/debian/ jessie/main libpsl0 amd64 0.5.1-1 [41.6 kB]
```

```
Get:11 http://httpredir.debian.org/debian/ jessie/main wget amd64 1.16-1 [495 kB]
debconf: delaying package configuration, since apt-utils is not installed
Fetched 8856 kB in 1s (8311 kB/s)
Selecting previously unselected package libgmp10:amd64.
(Reading database ... 7877 files and directories currently installed.)
Preparing to unpack .../libgmp10_2%3a6.0.0+dfsg-6_amd64.deb ...
Unpacking libgmp10:amd64 (2:6.0.0+dfsg-6) ...
Selecting previously unselected package libnettle4:amd64.
Preparing to unpack .../libnettle4_2.7.1-5_amd64.deb ...
Unpacking libnettle4:amd64 (2.7.1-5) ...
Selecting previously unselected package libhogweed2:amd64.
Preparing to unpack .../libhogweed2_2.7.1-5_amd64.deb ...
Unpacking libhogweed2:amd64 (2.7.1-5) ...
Selecting previously unselected package libffi6:amd64.
Preparing to unpack .../libffi6_3.1-2+b2_amd64.deb ...
Unpacking libffi6:amd64 (3.1-2+b2) ...
Selecting previously unselected package libp11-kit0:amd64.
Preparing to unpack .../libp11-kit0_0.20.7-1_amd64.deb ...
Unpacking libp11-kit0:amd64 (0.20.7-1) ...
Selecting previously unselected package libtasn1-6:amd64.
Preparing to unpack .../libtasn1-6_4.2-3+deb8u1_amd64.deb ...
Unpacking libtasn1-6:amd64 (4.2-3+deb8u1) ...
Selecting previously unselected package libgnutls-deb0-28:amd64.
Preparing to unpack .../libgnutls-deb0-28_3.3.8-6+deb8u3_amd64.deb ...
Unpacking libgnutls-deb0-28:amd64 (3.3.8-6+deb8u3) ...
Selecting previously unselected package libidn11:amd64.
Preparing to unpack .../libidn11_1.29-1+b2_amd64.deb ...
Unpacking libidn11:amd64 (1.29-1+b2) ...
Selecting previously unselected package libicu52:amd64.
Preparing to unpack .../libicu52_52.1-8+deb8u3_amd64.deb ...
Unpacking libicu52:amd64 (52.1-8+deb8u3) ...
Selecting previously unselected package libpsl0:amd64.
Preparing to unpack .../libpsl0_0.5.1-1_amd64.deb ...
Unpacking libpsl0:amd64 (0.5.1-1) ...
Selecting previously unselected package wget.
Preparing to unpack .../archives/wget_1.16-1_amd64.deb ...
Unpacking wget (1.16-1) ...
Setting up libgmp10:amd64 (2:6.0.0+dfsg-6) ...
Setting up libnettle4:amd64 (2.7.1-5) ...
Setting up libhogweed2:amd64 (2.7.1-5) ...
Setting up libffi6:amd64 (3.1-2+b2) ...
Setting up libp11-kit0:amd64 (0.20.7-1) ...
Setting up libtasn1-6:amd64 (4.2-3+deb8u1) ...
Setting up libgnutls-deb0-28:amd64 (3.3.8-6+deb8u3) ...
Setting up libidn11:amd64 (1.29-1+b2) ...
Setting up libicu52:amd64 (52.1-8+deb8u3) ...
Setting up libpsl0:amd64 (0.5.1-1) ...
Setting up wget (1.16-1) ...
Processing triggers for libc-bin (2.19-18+deb8u1) ...
 ---> 07b5eedd4306
Removing intermediate container 04ab2d00fd1c
Step 6 : RUN wget https://github.com/jwilder/docker-gen/releases/download/0.4.3/docker-gen-linux-amd64-0.4.3.tar.gz
 ---> Running in 8ca1d7e681a6
converted 'https://github.com/jwilder/docker-gen/releases/download/0.4.3/docker-gen-linux-amd64-0.4.3.tar.gz'
(ANSI_X3.4-1968) -> 'https://github.com/jwilder/docker-gen/releases/download/0.4.3/docker-gen-linux-amd64-
0.4.3.tar.gz' (UTF-8)
--2015-11-23 16:37:16--  https://github.com/jwilder/docker-gen/releases/download/0.4.3/docker-gen-linux-amd64-
0.4.3.tar.gz
Resolving github.com (github.com)... 192.30.252.131
Connecting to github.com (github.com)|192.30.252.131|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github-cloud.s3.amazonaws.com/releases/17762549/5f57e9a4-7644-11e5-873a-2c2c718eb174.gz?X-Amz-
Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAISTNZFOVBIJMK3TQ%2F20151123%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-
Date=20151123T163716Z&X-Amz-Expires=300&X-Amz-
Signature=1b9fa68e1cbf53e32bfefe515dca03e4617a971d2d71c5ffd998d55381424871&X-Amz-
SignedHeaders=host&actor_id=0&response-content-disposition=attachment%3B%20filename%3Ddocker-gen-linux-amd64-
0.4.3.tar.gz&response-content-type=application%2Foctet-stream [following]
converted 'https://github-cloud.s3.amazonaws.com/releases/17762549/5f57e9a4-7644-11e5-873a-2c2c718eb174.gz?X-Amz-
Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAISTNZFOVBIJMK3TQ%2F20151123%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-
Date=20151123T163716Z&X-Amz-Expires=300&X-Amz-
Signature=1b9fa68e1cbf53e32bfefe515dca03e4617a971d2d71c5ffd998d55381424871&X-Amz-
SignedHeaders=host&actor_id=0&response-content-disposition=attachment%3B%20filename%3Ddocker-gen-linux-amd64-
0.4.3.tar.gz&response-content-type=application%2Foctet-stream' (ANSI_X3.4-1968) -> 'https://github-
cloud.s3.amazonaws.com/releases/17762549/5f57e9a4-7644-11e5-873a-2c2c718eb174.gz?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-
Amz-Credential=AKIAISTNZFOVBIJMK3TQ/20151123/us-east-1/s3/aws4_request&X-Amz-Date=20151123T163716Z&X-Amz-
Expires=300&X-Amz-Signature=1b9fa68e1cbf53e32bfefe515dca03e4617a971d2d71c5ffd998d55381424871&X-Amz-
SignedHeaders=host&actor_id=0&response-content-disposition=attachment; filename=docker-gen-linux-amd64-
0.4.3.tar.gz&response-content-type=application/octet-stream' (UTF-8)
--2015-11-23 16:37:16--  https://github-cloud.s3.amazonaws.com/releases/17762549/5f57e9a4-7644-11e5-873a-
2c2c718eb174.gz?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAISTNZFOVBIJMK3TQ/20151123/us-east-
1/s3/aws4_request&X-Amz-Date=20151123T163716Z&X-Amz-Expires=300&X-Amz-
Signature=1b9fa68e1cbf53e32bfefe515dca03e4617a971d2d71c5ffd998d55381424871&X-Amz-
SignedHeaders=host&actor_id=0&response-content-disposition=attachment;%20filename=docker-gen-linux-amd64-
0.4.3.tar.gz&response-content-type=application/octet-stream
Resolving github-cloud.s3.amazonaws.com (github-cloud.s3.amazonaws.com)... 54.231.113.243
Connecting to github-cloud.s3.amazonaws.com (github-cloud.s3.amazonaws.com)|54.231.113.243|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2827064 (2.7M) [application/octet-stream]
Saving to: 'docker-gen-linux-amd64-0.4.3.tar.gz'

     0K .......... .......... .......... .......... ..........  1%  228K 12s
    50K .......... .......... .......... .......... ..........  3%  678K 8s
   100K .......... .......... .......... .......... ..........  5%  342K 8s
```

```
   150K .......... .......... .......... .......... ..........  7%  671K 7s
   200K .......... .......... .......... .......... ..........  9%  688K 6s
   250K .......... .......... .......... .......... .......... 10% 21.5M 5s
   300K .......... .......... .......... .......... .......... 12%  695K 5s
   350K .......... .......... .......... .......... .......... 14% 85.6M 4s
   400K .......... .......... .......... .......... .......... 16%  695K 4s
   450K .......... .......... .......... .......... .......... 18%  107M 3s
   500K .......... .......... .......... .......... .......... 19% 10.1M 3s
   550K .......... .......... .......... .......... .......... 21%  733K 3s
   600K .......... .......... .......... .......... .......... 23% 79.7M 3s
   650K .......... .......... .......... .......... .......... 25% 11.5M 2s
   700K .......... .......... .......... .......... .......... 27%  733K 2s
   750K .......... .......... .......... .......... .......... 28% 27.3M 2s
   800K .......... .......... .......... .......... .......... 30%  101M 2s
   850K .......... .......... .......... .......... .......... 32% 7.06M 2s
   900K .......... .......... .......... .......... .......... 34%  767K 2s
   950K .......... .......... .......... .......... .......... 36% 89.5M 2s
  1000K .......... .......... .......... .......... .......... 38% 95.3M 2s
  1050K .......... .......... .......... .......... .......... 39% 7.66M 1s
  1100K .......... .......... .......... .......... .......... 41%  754K 1s
  1150K .......... .......... .......... .......... .......... 43% 81.9M 1s
  1200K .......... .......... .......... .......... .......... 45%  104M 1s
  1250K .......... .......... .......... .......... .......... 47%  145M 1s
  1300K .......... .......... .......... .......... .......... 48% 1.30M 1s
  1350K .......... .......... .......... .......... .......... 50% 1.45M 1s
  1400K .......... .......... .......... .......... .......... 52%  129M 1s
  1450K .......... .......... .......... .......... .......... 54% 18.6M 1s
  1500K .......... .......... .......... .......... .......... 56%  135M 1s
  1550K .......... .......... .......... .......... .......... 57% 1.35M 1s
  1600K .......... .......... .......... .......... .......... 59% 1.45M 1s
  1650K .......... .......... .......... .......... .......... 61% 33.4M 1s
  1700K .......... .......... .......... .......... .......... 63%  137M 1s
  1750K .......... .......... .......... .......... .......... 65% 8.99M 1s
  1800K .......... .......... .......... .......... .......... 67% 41.0M 1s
  1850K .......... .......... .......... .......... .......... 68% 1.45M 1s
  1900K .......... .......... .......... .......... .......... 70% 1.50M 1s
  1950K .......... .......... .......... .......... .......... 72% 32.0M 0s
  2000K .......... .......... .......... .......... .......... 74% 97.5M 0s
  2050K .......... .......... .......... .......... .......... 76% 10.0M 0s
  2100K .......... .......... .......... .......... .......... 77% 25.1M 0s
  2150K .......... .......... .......... .......... .......... 79% 1.53M 0s
  2200K .......... .......... .......... .......... .......... 81% 1.52M 0s
  2250K .......... .......... .......... .......... .......... 83% 40.7M 0s
  2300K .......... .......... .......... .......... .......... 85%  103M 0s
  2350K .......... .......... .......... .......... .......... 86% 11.2M 0s
  2400K .......... .......... .......... .......... .......... 88% 12.5M 0s
  2450K .......... .......... .......... .......... .......... 90% 1.69M 0s
  2500K .......... .......... .......... .......... .......... 92% 1.46M 0s
  2550K .......... .......... .......... .......... .......... 94% 25.6M 0s
  2600K .......... .......... .......... .......... .......... 95% 23.2M 0s
  2650K .......... .......... .......... .......... .......... 97%  145M 0s
  2700K .......... .......... .......... .......... .......... 99% 12.4M 0s
  2750K ..........                                            100% 6.51M=1.4s

2015-11-23 16:37:18 (1.92 MB/s) - 'docker-gen-linux-amd64-0.4.3.tar.gz' saved [2827064/2827064]

 ---> 47487a78cbdf
Removing intermediate container 8ca1d7e681a6
Step 7 : RUN tar xvzf docker-gen-linux-amd64-0.4.3.tar.gz
 ---> Running in ccd93cea65bc
docker-gen
 ---> 6419da60ac4d
Removing intermediate container ccd93cea65bc
Step 8 : RUN chmod +x docker-gen
 ---> Running in 4555ba316fbc
 ---> b1a01d74bfe4
Removing intermediate container 4555ba316fbc
Step 9 : RUN mv docker-gen /opt/bin
 ---> Running in 133487c98874
 ---> 7b587a0403df
Removing intermediate container 133487c98874
Step 10 : ADD assets/nginx.tmpl /template/nginx.tmpl
 ---> ae6c1bcfb6db
Removing intermediate container 0e4b462194ee
Step 11 : ADD assets/nginx.conf /etc/nginx/nginx.conf
 ---> a528697661f1
Removing intermediate container fc4239847126
Step 12 : ADD assets/run.sh /opt/bin/run.sh
 ---> fa24e7027e3e
Removing intermediate container 682683e4e4db
Step 13 : RUN chmod +x /opt/bin/run.sh
 ---> Running in d3ab24f9d738
 ---> 3e7ce232c8b9
Removing intermediate container d3ab24f9d738
Step 14 : CMD /opt/bin/run.sh
 ---> Running in 52f0ce963d72
 ---> cb7f7245d9d1
Removing intermediate container 52f0ce963d72
Successfully built cb7f7245d9d1
```

Referències

1 - Docker: https://www.docker.com/

2 - Machine: https://docs.docker.com/machine/

3 - Swarm: https://docs.docker.com/swarm/

4 - Digital Ocean: https://www.digitalocean.com/

5 - CoreOS: https://coreos.com/

6 - NGINX: http://nginx.org/

7 - Spring Boot: http://projects.spring.io/spring-boot/

8 - BitBucket: https://bitbucket.org/

9 - DockerHub: https://hub.docker.com/

10 - jMeter: http://jmeter.apache.org/

11 - Docker-Gen: https://github.com/jwilder/docker-gen