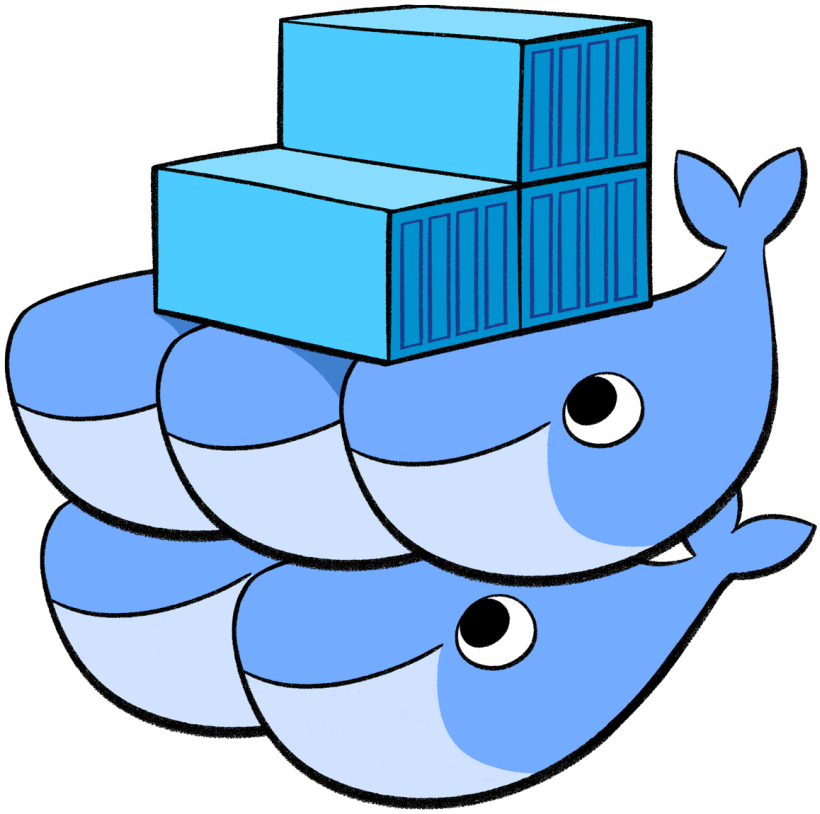
IES GONZALO NAZARENO

Proyecto Fin de Grado ASIR



Despliegue de una aplicación sobre Docker Swarm

Alberto Andrades Gil

Índice de contenido

Objetivo a conseguir

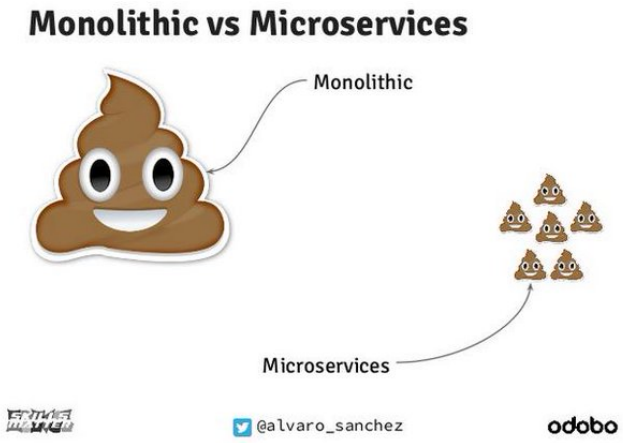
El objetivo de este proyecto es desplegar Drupal 8 en alta disponibilidad sobre contenedores Docker en un cluster Swarm. Esto conlleva mantener un sistema redundante, o también conocidos como FailOver, a cumplir con cada microservicio.

Porqué Drupal 8 y Docker Swarm

Elegí Drupal 8 al ser un CMS muy completo y adaptable a cualquier tipo de necesidad, ya sea un modo de negocio o fines educativos.

Me decanté a usar Docker Swarm para este proyecto por ser un entorno desplegable y escalable en corto tiempo, pudiendo así ya tener imágenes de cada microservicio creadas, y solo tener que lanzarlas y comenzar a balancear la carga de trabajo contra este nuevo contenedor.

El usar contenedores con cada microservicio me ofrece la ventaja de poder monitorizar cada servicio, detectar cuando está caído o detectar cuando se recibe una alta demanda de peticiones y poder escalar el servicio necesario.

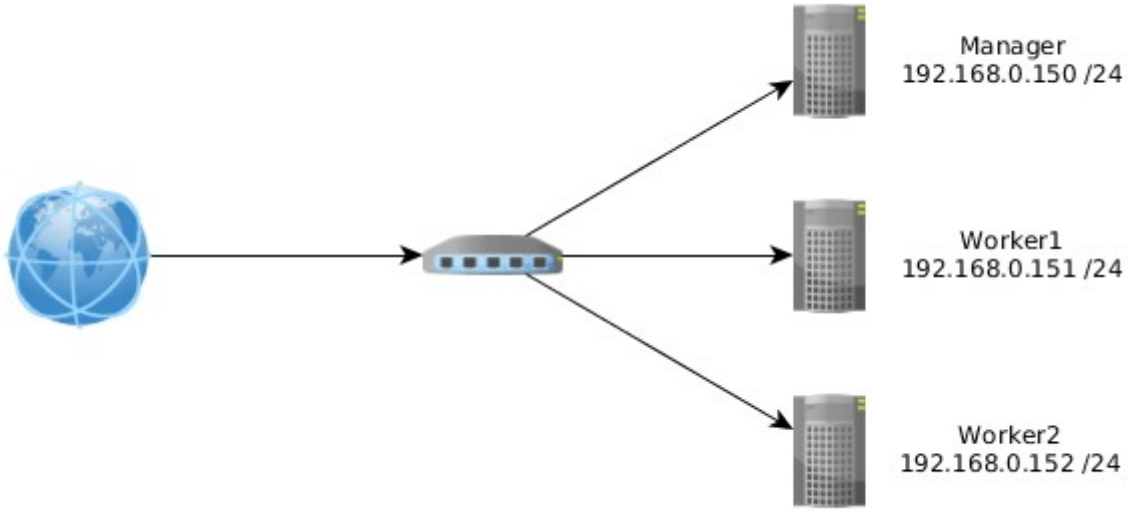


Todo esto me motiva al aprendizaje de un despliegue automatizado, la sincronización de diferentes servicios para hacer funcionar una aplicación completa, y sobre todo el saber mantener un sistema tolerante a fallos, flexible y adaptable a la carga de trabajo en cada momento.

Descripción del entorno:

Se presentan 3 máquinas virtuales sobre KVM con Debian Stretch como sistema anfitrión y también como huésped. Todas con 2 cores y 4GiB de RAM, 1 tarjeta de red con acceso a internet y que permite la conexión entre los 3 nodos.

- MV Manager: 192.168.0.150/24
- MV Worker1: 192.168.0.151/24
- MV Worker2: 192.168.0.152/24



Descripción de los componentes a usar:

GlusterFS

GlusterFS comparte el almacenamiento sobre la red, permitiendo escalar la capacidad o replicar el almacenamiento en otro nodo y aumentar la velocidad de acceso al sistema.

El almacenamiento a compartir es para Apache2 y PHP7.0, ya que en cada contenedor se necesita el acceso a los mismos ficheros, ofreciendo así siempre el contenido actualizado, sin retraso por sincronizaciones. Este espacio se configurará en el nodo Manager, y desde los Workers se accede para después ser ofrecido a los contenedores.

Varnish

Varnish permite almacenar en memoria caché las peticiones ya servidas, por lo que acelera la respuesta de la aplicación web. en la memoria o en el disco duro.

Ofrece su servicio estando delante del servidor web, siendo este caso Apache2.

Apache2.4

Apache se encarga de servir principalmente contenido web estático, aunque en este caso, mediante un proxy fastcgi, se comunica con el procesador PHP 7.0 para servir contenido dinámico.

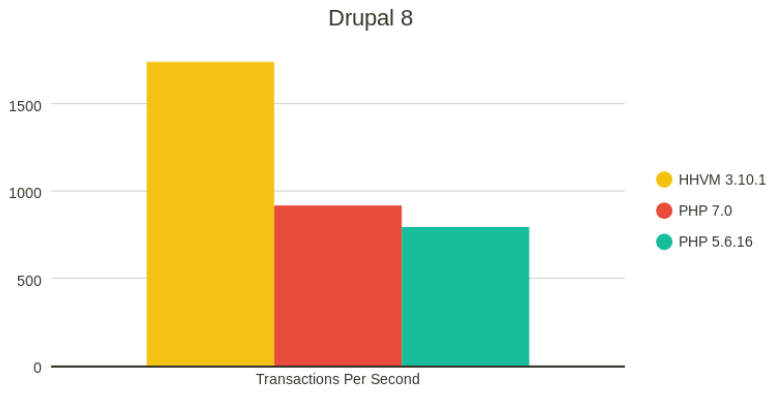
En Debian Jessie, era necesario instalar el paquete `libapache2-mod-fastcgi` para conectar Apache con PHP, en cambio, en Debian Stretch ya no es necesario, se puede utilizar el módulo `proxy_fcgi` e indicar si el listener de PHP es mediante un socket del sistema o IP.

PHP 7.0 + FPM

PHP7.0 es la última versión disponible del procesador de código PHP en Debian Strech.

FPM es la versión mejorada de FastCGI, gestionando mejor los procesos y utilizando la memoria RAM como caché de recursos.

Hoy en día existe un rival bastante duro para este procesador. Se conoce como HHVM (HipHop Virtual Machine), y es más eficiente que PHP5 y en algunos casos, más que PHP7.0.



Memcached

Memcached ofrece una memoria caché, registrando las peticiones a la base de datos y acelerando así cualquier procesamiento que requiera lectura de la base de datos, consiguiendo también reducir la carga a la base de datos.

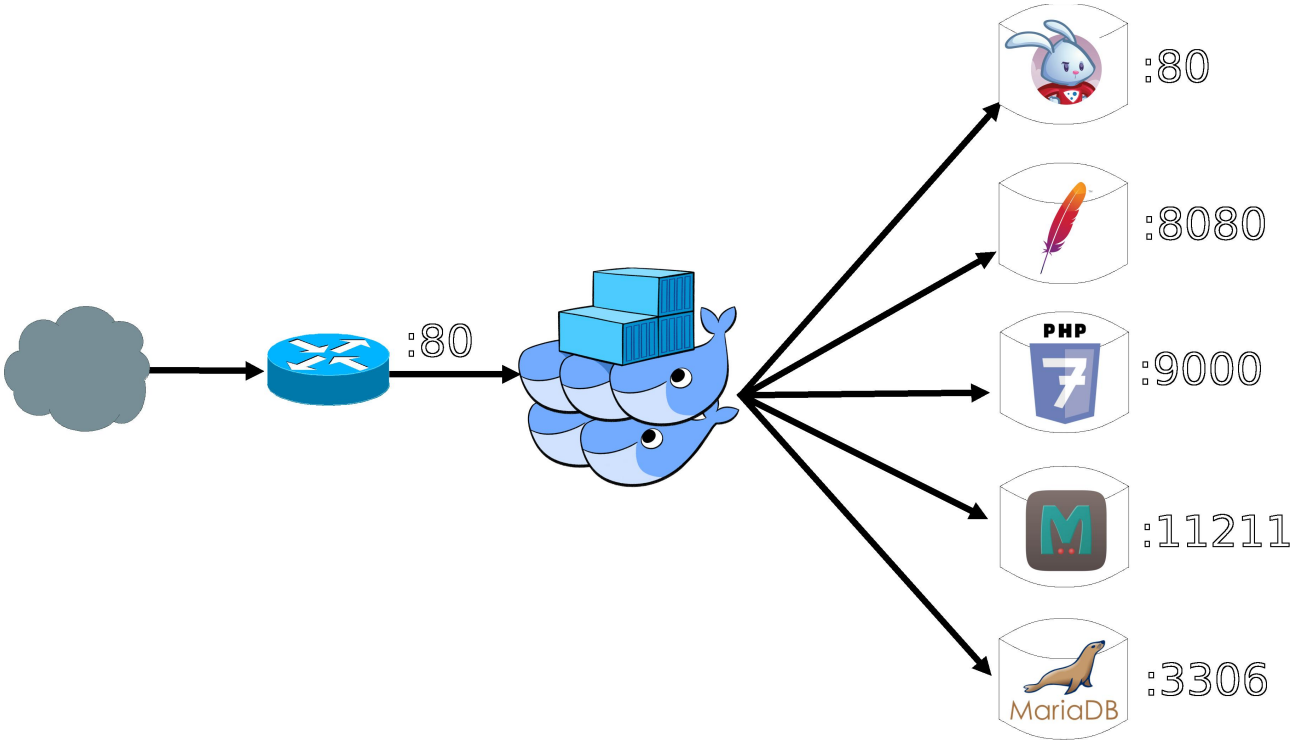
Galera Cluster

Galera Cluster es un conjunto de gestores de bases de datos MariaDB, sincronizados entre si, permitiendo estar balanceando carga en modo activo-activo, baja latencia en la sincronización de registros, sin perdida de transacciones, y escalable para conseguir un mayor número de peticiones.

Diagrama de funcionamiento

El desarrollo de una petición se trataría de la siguiente forma teniendo en cuenta que Docker Swarm balancea la carga por puertos:

1. Un cliente solicita el portal ofrecido por Drupal 8
2. Docker Swarm recibe la petición y la transmite a un Varnish disponible en el puerto 80
3. Varnish trata la petición y si no está cacheada, la traslada a Apache por el puerto 8080.
4. Apache se comunica con el procesador PHP a través del puerto 9000
5. PHP solicita a Memcached la caché a través del puerto 11211
6. PHP consulta la base de datos Galera al puerto 3306
7. Una vez realizado todo este procedimiento, se responde al cliente con la petición completamente generada.



Preparando Docker para el cluster:

Para el despliegue de un cluster Swarm necesitamos tener previamente instalado Docker en los tres nodos del cluster.

La instalación se realizará desde el repositorio de Docker, y para usarlo necesitamos instalar las siguiente dependencias:

```
sudo apt install apt-transport-https ca-certificates curl
software-properties-common -y
```

Añadimos su repositorio y la firma GPG:

```
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo
apt-key add -

sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/debian $(lsb_release -cs)
stable"
```

Actualizamos los repositorios e instalamos docker-ce:

```
apt update && apt install docker-ce -y
```

Creación del cluster y agregación de nodos

Una vez instalado docker en los tres nodos, elegimos uno que sea el nodo que gobierne a los otros nodos:

Sobre el nodo manager:

```
docker swarm init --advertise-addr 192.168.0.150
```

La IP asociada al parámetro advertise-addr debe ser la dirección por la cual se orquestarán los servicios que albergue el cluster, no importando que sea pública o privada.

Al ejecutar la creación del cluster Swarm, éste nos devuelve un comando con un token para unir nodos al cluster.

```
root@manager:~# docker swarm init --advertise-addr 192.168.0.150
Swarm initialized: current node (q9q4h9wwrc78wevnwcl6i2pbj) is now
a manager.

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

    docker swarm join \
    --token SWMTKN-1-
2g53ieqpdvif1poo8s85f7368elz2ta39sw201j7z186qused9-
abpi0t1y9jdu1ldtlqkx18wge \
    192.168.0.150:2377
```

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

Añadimos **worker1** y **worker2** al cluster con el comando y el token ofrecido:

```
root@worker1:~#     docker swarm join \
>     --token SWMTKN-1-
2g53ieqpdvif1poo8s85f7368elz2ta39sw201j7z186qused9-
abpi0t1y9jdu1ldtlqkx18wge \
>     192.168.0.150:2377
This node joined a swarm as a worker.
```

```
root@worker2:~#     docker swarm join \
>     --token SWMTKN-1-
2g53ieqpdvif1poo8s85f7368elz2ta39sw201j7z186qused9-
abpi0t1y9jdu1ldtlqkx18wge \
>     192.168.0.150:2377
This node joined a swarm as a worker.
```

Si nos muestra el siguiente error, puede ser debido a un error tipográfico al escribir la IP del manager al crear el cluster o al unir un nodo al mismo.

```
Error response from daemon: rpc error: code = 14 desc = grpc: the
connection is unavailable
```

Desde el manager podemos ver información del cluster con `docker info`:

```
root@manager:~# docker info
. . .
Swarm: active
 NodeID: q9q4h9wwrc78wevnwcl6i2pbj
 Is Manager: true
 ClusterID: uglxa5i9dd1p73lk6qibu4f2u
 Managers: 1
 Nodes: 3
. . .
 Node Address: 192.168.0.150
 Manager Addresses:
  192.168.0.150:2377
. . .
```

Podemos ver el conjunto de nodos en el cluster con `docker node ls`:

```
root@manager:~# docker node ls
ID                           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
q9q4h9wwrc78wevnwcl6i2pbj *  manager   Ready   Active        Leader
svm9buugd57fuiloh4206aw9e    worker2   Ready   Active
xbs40nejtycw9q4mc8kfortnb    worker1   Ready   Active
```

Instalación y configuración de GlusterFS sobre los nodos

GlusterFS Server se instalará en el nodo Manager y compartirá un directorio donde se alojará todo el contenido de Drupal 8, y este será ofrecido a los nodos Workers que a su vez montarán como volumen los contenedores de los servicios Apache2 y PHP.

Para instalar GlusterFS en el nodo Manager:

```
apt install glusterfs-server -y
```

Se crea el directorio y el volumen que será compartido mediante GlusterFS:

```
mkdir /gluster_vol
gluster volume create apache-fpm-vol \
 transport tcp 192.168.0.150:/gluster_vol force
```

Una vez creado el volumen, se debe iniciar:

```
gluster volume start apache-fpm-vol
```

Para tener la misma ruta de acceso que los workers, se puede crear un enlace simbólico:

```
ln -s /gluster_vol /mnt/apache-fpm-vol
```

Instalamos en los nodos Workers el cliente de GlusterFS:

```
apt install glusterfs-client -y
```

Creamos el directorio donde se montará el volumen, añadimos al fichero fstab el automontaje del volumen y comprobamos que se ha montado correctamente:

```
mkdir /mnt/apache-fpm-vol

echo "192.168.0.150:/apache-fpm-vol /mnt/apache-fpm-vol glusterfs
defaults,_netdev 0 0" >> /etc/fstab

mount -a
```

Creación de las imágenes de cada microservicio

Se creará una imagen por cada microservicio que será compartida en Docker Hub, y posteriormente cuando se despliega el Stack sobre Swarm se descargará y ejecutará.

Contenido común de cada Dockerfile:

En cada Dockerfile se han especificado las siguientes directivas para el contenedor:

- Sistema operativo a utilizar
- Persona encargada del Dockerfile
- Actualización completa del sistema operativo
- Instalación del software a utilizar
- Ficheros de configuración del software
- Limpieza del contenedor (archivos generados durante la instalación de dependencias)
- Puerto por el cual se obtendrán los servicios del contenedor
- Límite de memoria RAM que usará
- Usuario que ejecutará el proceso del servicio
- Proceso o script a ejecutar en el contenedores

Varnish

Los paquetes a instalar son:

- `Varnish`
- `dnsutils`

El fichero de configuración que utiliza es para definir la IP del balanceador de carga, redireccionando así el tráfico a los servidores web.

El script sustituye la IP que se le proporciona mediante variable e inicia el proceso de Varnish.

Apache

Los paquetes a instalar son:

- `apache2`
- `wget`
- `dnsutils`

Se habilitan los módulos necesarios, se habilita el nuevo sitio web que se proporciona mediante fichero de configuración y el script sustituye el puerto del listener al 8080, descarga el CMS e inicia el proceso de Apache.

PHP 7.0 + FPM

Los paquetes esenciales a instalar son:

- `php7.0`
- `php7.0-fpm`
- Dependencias de Drupal 8

El fichero de configuración modifica el listener de socket a IP y el script sustituye la ip del listener e inicia el proceso de PHP7.0

Memcached

El único paquete a instalar es `memcached`.

Galera Cluster

Para este componente, he usado la imagen del siguiente repositorio.

Se parte de un único contenedor y al escalar reconoce los nodos de la base de datos y se sincroniza, manteniendo así la base de datos en alta disponibilidad.

Proceso de creación de imágenes

Todo el código de este proyecto se encuentra actualizado en el siguiente repositorio:

https://github.com/traskiloner/drupal-microservices

Para crear la imagen y subirla al repositorio es necesario primero tener una cuenta en Docker Hub.

Nos identificamos en Docker Hub desde una terminal:

```
docker login
```

Creamos la imagen mediante el Dockerfile:

```
docker build -t drupalmicroservices_apache .
```

Asociamos una etiqueta a la imagen creada para subirla al repositorio

```
docker tag drupalmicroservices_apache
traskiloner/drupal8:apache_v1
```

Subimos la imagen al repositorio:

```
docker push traskiloner/drupal8
```

Despliegue de la aplicación

Para realizar el despliegue de la aplicación, decidí realizar un Stack de servicios, consiguiendo así poder reunir todos los servicios necesarios con sus parámetros, en un documento en formato Yaml.

Creación del stack

El Stack a crear reunirá los servicios y el volumen que se asociará a los contenedores, así como los parámetros para la base de datos y el número de réplicas de cada servicio.

```
---
version: "3.0"

services:

  varnish:
    image: traskiloner/drupal8:varnish_v2
    ports:
      - "80:80"
    deploy:
      replicas: 1

  apache:
    image: traskiloner/drupal8:apache_v2
    volumes:
      - "apache-fpm:/var/www"
    deploy:
      replicas: 1

  phpfpm:
    image: traskiloner/drupal8:phpfpm_v2
    volumes:
      - "apache-fpm:/var/www"
    deploy:
      replicas: 1

  memcached:
    image: traskiloner/drupal8:memcached_v2

  galera:
    image: toughiq/mariadb-cluster
    deploy:
      replicas: 1
    environment:
      - DB_SERVICE_NAME=galera
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_DATABASE=drupaldb
      - MYSQL_USER=dbuser
      - MYSQL_PASSWORD=dbpassword

volumes:
  apache-fpm:
    driver: local
    driver_opts:
     o: bind
     type: none
     device: /mnt/apache-fpm-vol
...
```

Despliegue del stack

Una vez generado el stack, procedemos al despliegue automatizado de los servicios desde el nodo manager, siendo este quien reparte los contenedores a cada worker:

```
root@manager:~# docker stack deploy -c stack.yml drupal8
Creating network drupal8_default
Creating service drupal8_galera
Creating service drupal8_varnish
Creating service drupal8_apache
Creating service drupal8_phpfpm
Creating service drupal8_memcached
```

El tiempo de despliegue depende de la carga de trabajo de cada nodo.

Podemos comprobar el estado del stack y ver las replicas de cada servicio:

```
docker service ls
```

Para ver donde se está ejecutando cada servicio:

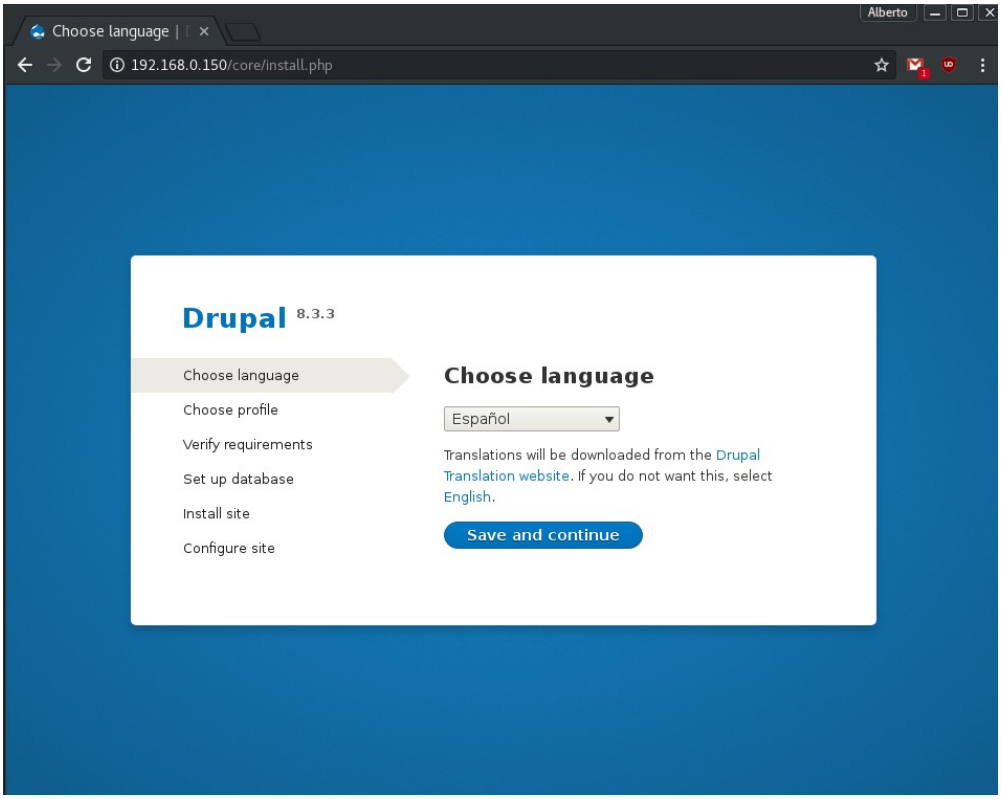
```
docker service ps drupal8_apache
```

Para escalar el número de contenedores de un servicio:

```
docker service scale drupal8_apache=2
```

Prueba de funcionamiento

Al acceder a la IP del manager, podemos visualizar el portal de Drupal:

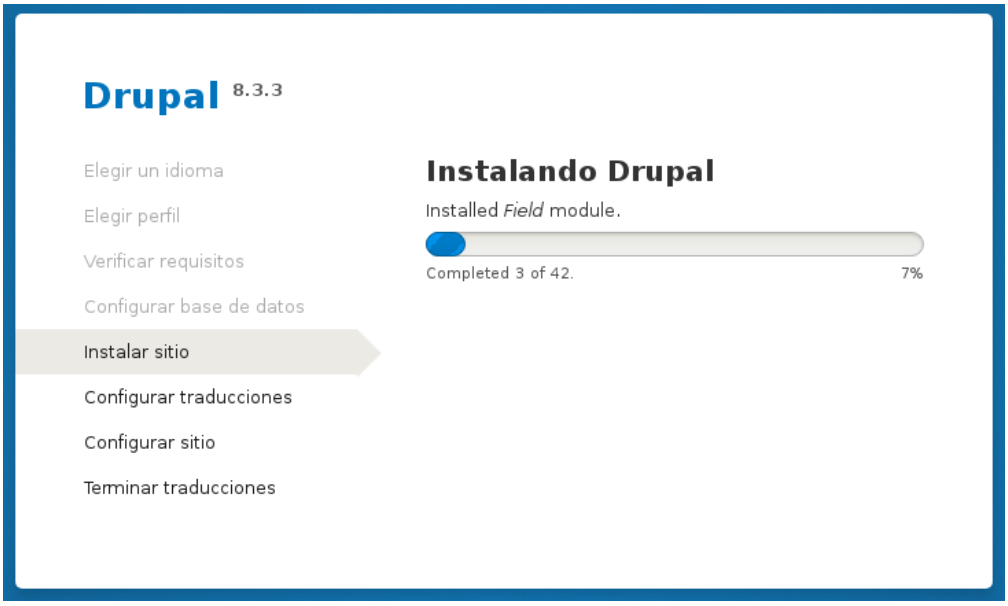


Se puede ver el log de acceso accediendo al contenedor de apache:

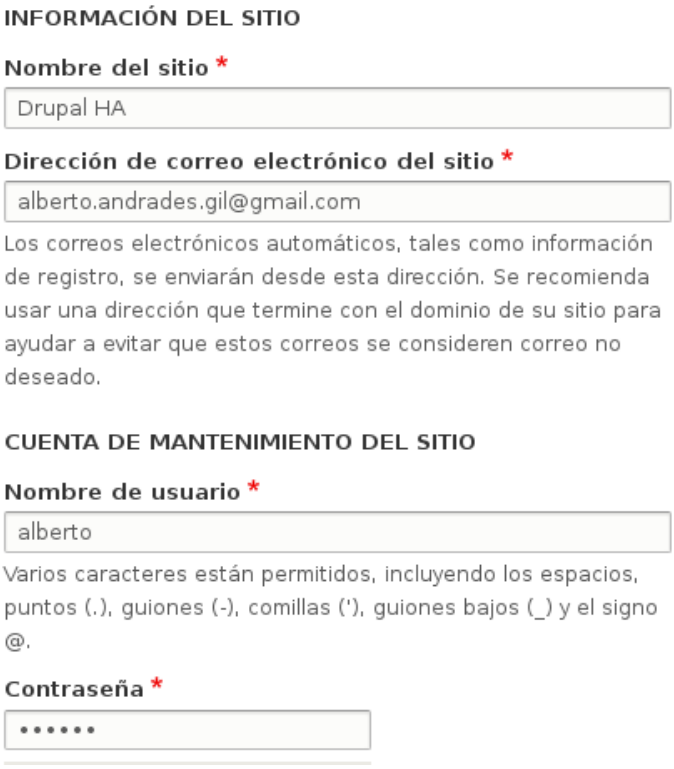
```
root@worker1:~# docker exec -it
drupal8_apache.1.pjg5cnhfd2hti44gyjvqlvm3o bash

root@7631c79939d5:/# tailf /var/log/apache2/access.log
10.0.0.5 - - [16/Jun/2017:09:25:22 +0000] "POST /core/install.php?
langcode=es&profile=standard&continue=1&id=1&op=do_nojs&op=do&_for
mat=json HTTP/1.1" 200 525 "http://192.168.0.150/core/install.php?
langcode=es&profile=standard&continue=1&id=1&op=start"
"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/59.0.3071.86 Safari/537.36"
```

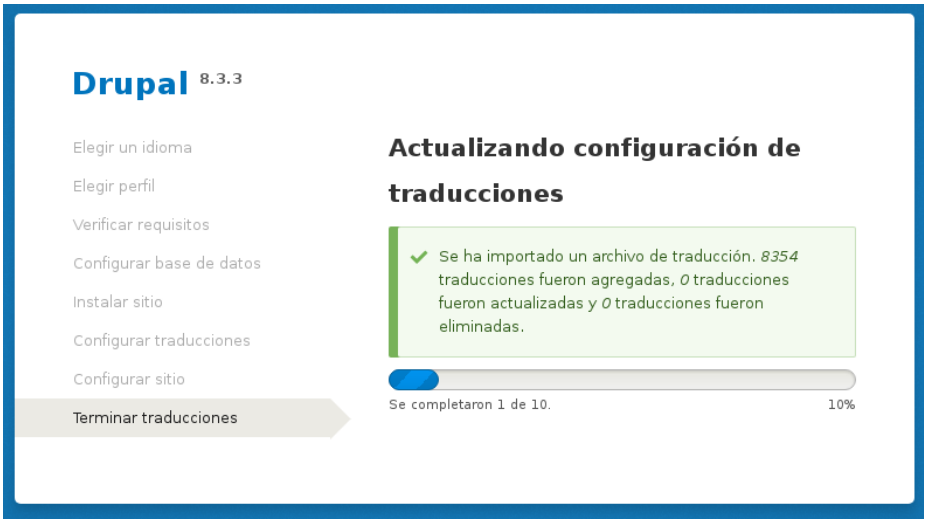
Se introducen los valores de la base de datos y comienza su instalación:



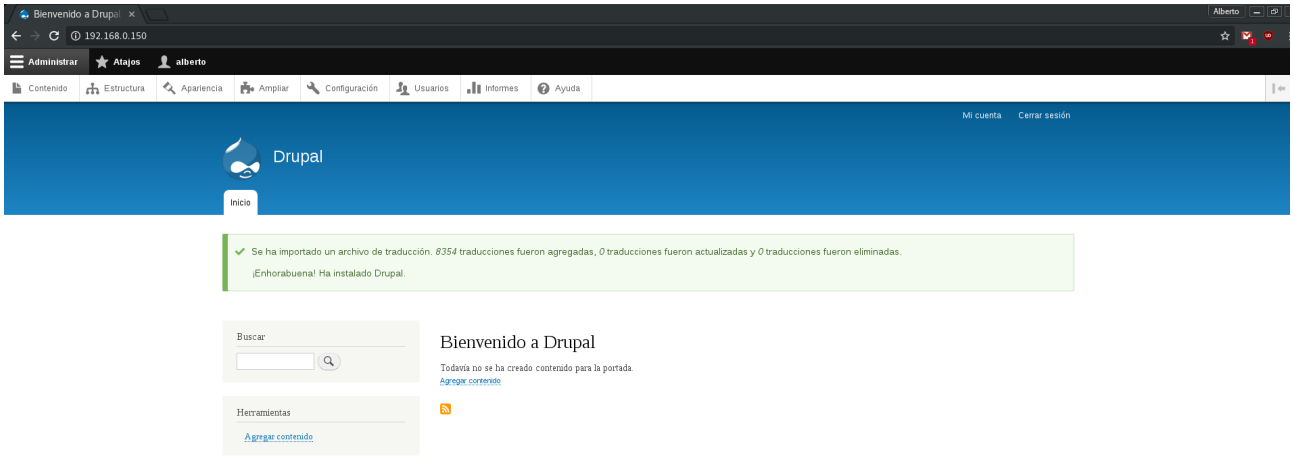
Introducimos el nombre del sitio y un usuario administrador:



Y termina la configuración de Drupal 8:



Ya está accesible Drupal 8:



Pruebas de alta disponibilidad

Para comprobar que la aplicación tiene tolerancia a fallos, se puede escalar cada contenedor y observar como se replica el servicio sin provocar perdidas ni caidas de servicio en caso de fallo:

```
root@manager:~# docker service scale drupal8_apache=2
drupal8_phpfpm=2 drupal8_varnish=2

drupal8_apache scaled to 2
drupal8_phpfpm scaled to 2
drupal8_varnish scaled to 2
```

Se comprueba que se han duplicado los indicados:

```
root@manager:~# docker service ls
ID            NAME               MODE         REPLICAS   IMAGE
6gq70nihz9ve  drupal8_apache     replicated   2/2        traskiloner/drupal8:apache_v2
invwlb8n5wwe  drupal8_galera     replicated   1/1        toughiq/mariadb-cluster:latest
vczn9vzzcedp  drupal8_varnish    replicated   2/2        traskiloner/drupal8:varnish_v2
wlnjulcx0xwp  drupal8_phpfpm     replicated   2/2        traskiloner/drupal8:phpfpm_v2
zp9f2c9dk4kd  drupal8_memcached  replicated   1/1        traskiloner/drupal8:memcached_v2
```

Paramos un contenedor de apache para comprobar que Swarm vuelve a lanzar un contenedor y mantiene dos replicas:

```
root@worker1:~# docker stop 7631c79939d5
```

```
root@manager:~# docker service ps drupal8_apache
ID            NAME                  IMAGE
NODE     DESIRED STATE  CURRENT STATE               ERROR
PORTS

qmz4ue2v817a  drupal8_apache.1      traskiloner/drupal8:apache_v2
worker1  Running Running about a minute ago

pjg5cnhfd2ht   \_ drupal8_apache.1  traskiloner/drupal8:apache_v2
worker1  Shutdown       Failed about a minute ago   "task: non-
zero exit (137)"

lo5axeapk5b0  drupal8_apache.2      traskiloner/drupal8:apache_v2
worker2  Running        Running 3 minutes ago
```

Se observa el contenedor que fue detenido manualmente, el contenedor que se lanzó al elevar el número de replicas y el que lanzó Swarm al detectar la caida.

Prueba de sincronización en la base de datos

Para comprobar que la base de datos se sincroniza correctamente, se escala el servicio a 3 contenedores y se comprueba que se ha sincronizado la base de datos en los dos nuevos contenedores.

```
root@manager:~#  docker service scale drupal8_galera=3

drupal8_galera scaled to 3
```

Se comprueba que se ha escalado correctamente:

```
root@manager:~# docker service ls
ID            NAME               MODE        REPLICAS  IMAGE
6gq70nihz9ve  drupal8_apache     replicated  2/2       traskiloner/drupal8:apache_v2
invwlb8n5wwe  drupal8_galera     replicated  3/3       toughiq/mariadb-cluster:latest
vczn9vzzcedp  drupal8_varnish    replicated  2/2       traskiloner/drupal8:varnish_v2
wlnjulcx0xwp  drupal8_phpfpm     replicated  2/2       traskiloner/drupal8:phpfpm_v2
zp9f2c9dk4kd  drupal8_memcached  replicated  1/1       traskiloner/drupal8:memcached_v2
```

Comprobamos en que nodos se han desplegado las nuevas replicas:

```
root@manager:~# docker service ps drupal8_galera
ID            NAME              IMAGE NODE     DESIRED STATE
CURRENT STATE              ERROR  PORTS

5ktu2u608eyu  drupal8_galera.1  toughiq/mariadb-cluster:latest
manager  Running        Running about an hour ago

9ugrui8ztadx  drupal8_galera.2  toughiq/mariadb-cluster:latest
worker1  Running        Running 9 minutes ago

vge7uz9b970d  drupal8_galera.3  toughiq/mariadb-cluster:latest
worker2  Running        Running 9 minutes ago
```

Accedemos con un cliente mysql a cada gestor de base de datos y comprobamos que los registros se han sincronizado:

```
root@18d596747820:/# mysql -u dbuser -pdbpassword -D drupaldb
MariaDB [drupaldb]> select uid,name from users_field_data;
+-----+---------+
| uid | name    |
+-----+---------+
|   0 |         |
|   1 | alberto |
+-----+---------+
2 rows in set (0.00 sec)
```

```
root@4f856bfb70f6:/# mysql -u dbuser -pdbpassword -D drupaldb
MariaDB [drupaldb]> select uid,name from users_field_data;
+-----+---------+
| uid | name    |
+-----+---------+
|   0 |         |
|   1 | alberto |
+-----+---------+
2 rows in set (0.00 sec)
```

```
root@8d3741502a08:/# mysql -u dbuser -pdbpassword -D drupaldb
MariaDB [drupaldb]> select uid,name from users_field_data;
+-----+---------+
| uid | name    |
+-----+---------+
|   0 |         |
|   1 | alberto |
+-----+---------+
2 rows in set (0.00 sec)
```

Se ha replicado y sincronizado correctamente.

Enlaces de interés:

https://www.drupal.org/docs/8/install

http://www.davidam.com/docu/installingdrupal.html ← Learning Drush

http://docs.drush.org/en/master/install/

https://www.drupal.org/documentation/install/developers

http://bretfisher.com/10-minutes-to-highly-available-docker/

https://serverfault.com/questions/344171/apache2-fcgid-not-fastcgi-with-php-fpm ← Apache2.4 en Stretch no usa fastcgi, se usa el mod ya instalado con apache2 proxy_fcgi

https://httpd.apache.org/docs/2.4/mod/mod_proxy_fcgi.html ← Documentación mod_proxy_fcgi

https://docs.docker.com/engine/swarm/swarm-tutorial/#three-networked-host-machines ← Tipos de red en Docker Swarm

https://docs.docker.com/engine/reference/commandline/stack_deploy/#options ← Stack en Swarm

https://technologyconversations.com/2017/01/23/using-docker-stack-and-compose-yaml-files-to-deploy-swarm-services/ ← Caso práctico de Stack

https://docs.docker.com/engine/swarm/stack-deploy/#deploy-the-stack-to-the-swarm

https://github.com/traskiloner/drupal-microservices ← Repositorio actualizado