Centros de Datos y Provisión de Servicios

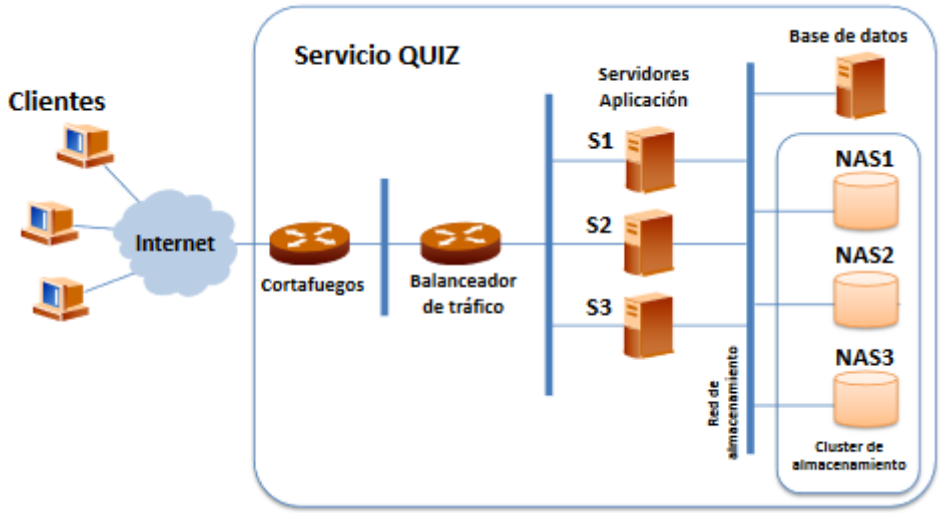
Práctica Final

Despliegue de una aplicación escalable

Luis Martín de Vidales Palomero
Verónica Santos Pulpón

Introducción

El objetivo de esta práctica es la implementación de la arquitectura completa para el despliegue del proyecto QUIZ utilizado en CORE. Para la arquitectura de despliegue utilizaremos elementos típicos de las arquitecturas actuales: firewall, balanceador de carga, servidores front-end, bases de datos y servidores de almacenamiento.



Esta aplicación se configurará para que utilice una base de datos, que correrá en el servidor destinado a ello. Por otra parte, las imágenes serán almacenadas en un cluster de almacenamiento formado por 3 servidores NAS. El balanceador de carga se ocupará de distribuir la carga entre los tres servidores y el cortafuegos de entrada se ocupará de filtrar todo el tráfico procedente de internet, dejando pasar únicamente el destinado a la aplicación.

Pasos seguidos en la implementación y despliegue del sistema

Primero se llevará a cabo la configuración del escenario de la práctica, para lo cual emplearemos el software VNX con el cual partiendo del fichero xml en el que se describe el escenario, levantaremos todas las máquinas de las que se compone, así como las redes entre ellas.

Una vez levantado el escenario, el orden de configuración de servicios seguido en la práctica es el siguiente:

- Configuración de la base de datos MariaDB
- Configuración del clúster de almacenamiento por medio del sistema de ficheros glusterfs.
- Configuración del servicio Quiz en los servidores front end. Para ello se emplean los paquetes nodejs, npm, forever y mySQL2
- Configuración del balanceador de carga empleando un balanceador de tipo haproxy.
- Configuración del firewall por medio del software firewall builder.

Posteriormente se implementan las mejoras:

- Script de configuración de un nuevo servidor front end
- Mejoras en el algoritmo de balanceo de carga

Puntos débiles de la arquitectura en cuanto a fiabilidad y escalabilidad y posibles soluciones.

En lo referente a la escalabilidad la dificultad a la hora de ampliar el clúster de almacenamiento, teniendo que eliminar el clúster y crear uno nuevo cambiando su configuración y el código del script es la debilidad principal.

No obstante, esa debilidad es fácilmente solucionable a través de la creación de un script que borrase el antiguo clúster, y solicitase el número de servidores a incluir como argumento, pasando a crearse un nuevo clúster con el número de servidores deseado. Para ello se debe disponer del número de servidores que se indique como argumento.

El principal problema en cuanto a fiabilidad es que existen puntos únicos de fallo en la base de datos y en el balanceador, de tal forma que si caen estos servidores se detiene por completo el servicio.

En lo referente a la base de datos, se podría implementar una base de datos que trabaje en modo espejo con la que hay desplegada, estableciendo una réplica como sucede con los nas para tener un respaldo en caso de error.

Para solucionar el caso del balanceador se podría añadir un segundo balanceador para redireccionar el tráfico hacia los servidores, e incluir una herramienta de administración como puede ser Puppet, en la cual definiríamos un estado en el cual funciona correctamente el balanceador 1, y si este dejase de funcionar recondujese el tráfico al balanceador 2 y realizase pruebas sobre el balanceador 1 y tratase de reparar la avería.

Configuración del escenario de la practica

Arrancamos el escenario de la practica descargándonos el fichero proporcionado y ejecutándolo con VirtualBox. Una vez hecho esto descargaremos y descomprimiremos el escenario proporcionado e iniciaremos el mismo con la orden **sudo vnx -f pfinal.xml –create.** Accedemos a las diferentes máquinas virtuales mediante SSH (**ssh root@s1**).

A continuación, modificaremos la imagen de las maquinas virtuales teniendo en cuenta que todas las máquinas virtuales utilizan el mismo sistema de ficheros. Realizaremos los siguientes pasos:

- Paramos el escenario de la practica: **sudo vnx -f pfinal.xml --destroy**
- Arrancamos una VM que utilice el modo directo y conectividad a internet la imagen de las VM: **sudo vnx --modify-rootfsfilesystems/rootfs_lxc64-cdps --arch=x86_64**
- Activamos la red: **sudo dhclient eth0**
- Por último, una vez instalado todo lo necesario, paramos la máquina virtual.

Después de todo esto ya podremos empezar a configurar los servicios.

Configuración de la Base de Datos

La base de datos que utilizaremos será MariaDB, que es un sistema de gestión de bases de datos derivado de MySQL, que correrá en el servidor BBDD y que podrá accederse en remoto desde los 3 servidores para almacenar la información de nuestra aplicación.

Para la automatización del despliegue creamos el script donde desarrollaremos los pasos necesarios para que se implemente lo comentado anteriormente. En este fichero también añadiremos una forma de comprobar que la base de datos esté funcionando.

```
sudo lxc-attach --clear-env -n bbdd -- apt update
sudo lxc-attach --clear-env -n bbdd -- apt -y install mariadb-server
sudo lxc-attach --clear-env -n bbdd -- sed -i -e 's/bind-address.*/bind-address=0.0.0.0/' -e 's/utf8mb4/utf8/' /etc/mysql/mariadb.conf.d/50-server.cnf
sudo lxc-attach --clear-env -n bbdd -- systemctl restart mysql
sudo lxc-attach --clear-env -n bbdd -- mysqladmin -u root password xxxx
sudo lxc-attach --clear-env -n bbdd -- mysql -u root --password='xxxx' -e "CREATE USER 'quiz' IDENTIFIED BY 'xxxx';"
sudo lxc-attach --clear-env -n bbdd -- mysql -u root --password='xxxx' -e "CREATE DATABASE quiz;"
sudo lxc-attach --clear-env -n bbdd -- mysql -u root --password='xxxx' -e "GRANT ALL PRIVILEGES ON quiz.* to 'quiz'@'localhost' IDENTIFIED by 'xxxx';"
sudo lxc-attach --clear-env -n bbdd -- mysql -u root --password='xxxx' -e "GRANT ALL PRIVILEGES ON quiz.* to 'quiz'@'%' IDENTIFIED by 'xxxx';"
sudo lxc-attach --clear-env -n bbdd -- mysql -u root --password='xxxx' -e "FLUSH PRIVILEGES;"

sudo lxc-attach --clear-env -n s1 -- apt -y install mariadb-client
sudo lxc-attach --clear-env -n s1 -- mysql -h 20.2.4.31 -u quiz --password='xxxx' quiz
```

Configuración del Gluster

Lo siguiente que configuraremos será el sistema de ficheros glusterfs en los servidores nas, siguiendo los pasos del enunciado a fin de conseguir alta disponibilidad. Para ello configuraremos los servidores nas de forma que estén redundados.

Creamos un script con los pasos a seguir para la configuración.

Primero añadiremos los servidores nas1, nas2 y nas3 al cluster ejecutando **gluster peer probe 20.4.2.22** para nas2 y **gluster peer probe 20.4.2.23** para nas3. Comprobamos su estado mediante **gluster peer status**.

A continuación, crearemos un volumen con los tres servidores que tienen que replicar la información **usando gluster volume create nas replica 3 nas1:/nas nas2:/nas nas3:/nas force**

Arrancamos y comprobamos el estado de estos volúmenes creados.

Para agilizar la recuperación del volumen ante caídas de los servidores cambiamos el valor del timeout en todos a 5 mediante la siguiente orden:

gluster volume set nas network.ping-timeout 5

Por último, crearemos un directorio en cada servidor donde montaremos el cluster, en el que se almacenarán las imágenes replicadas. La caída de un servidor se traduce en que los otros siguen actualizándose y este mantiene los archivos incluidos hasta el momento. Al volver a añadir el 3er nas este se sincroniza con los otros servidores de nuevo.

mkdir /mnt/nas

mount -t glusterfs 20.2.4.21:/nas /mnt/nas

Fichero configGluster.sh:

```
1
2   sudo lxc-attach --clear-env -n nas1 -- gluster peer probe 20.2.4.22
3
4   sudo lxc-attach --clear-env -n nas1 -- gluster peer probe 20.2.4.23
5
6   sudo lxc-attach --clear-env -n nas1 -- gluster peer status
7
8   sudo lxc-attach --clear-env -n nas1 -- gluster volume create nas replica 3 nas1:/nas nas2:/nas nas3:/nas force
9
10  sudo lxc-attach --clear-env -n nas1 -- gluster volume start nas
11
12  sudo lxc-attach --clear-env -n nas1 -- gluster volume info
13
14  sudo lxc-attach --clear-env -n nas1 -- gluster volume set nas network.ping-timeout 5
15
16  sudo lxc-attach --clear-env -n s1 -- mkdir /mnt/nas
17  sudo lxc-attach --clear-env -n s1 -- mount -t glusterfs 20.2.4.21:/nas /mnt/nas
18
19  sudo lxc-attach --clear-env -n s2 -- mkdir /mnt/nas
20  sudo lxc-attach --clear-env -n s2 -- mount -t glusterfs 20.2.4.21:/nas /mnt/nas
21
22  sudo lxc-attach --clear-env -n s3 -- mkdir /mnt/nas
23  sudo lxc-attach --clear-env -n s3 -- mount -t glusterfs 20.2.4.21:/nas /mnt/nas
24
```

Instalación y configuración del QUIZ

Nos instalamos lo primero node.js y npm en los tres servidores. Después clonaremos el repositorio del QUIZ y lo arrancaremos.

Lo que hemos modificado del código del enunciado es el comando mkdir public/uploads, que en vez de crear el repositorio directamente lo que hemos hecho es:

cd public;

ln -s /mnt/nas uploads;

Lo que hacemos con este comando es crearnos una carpeta uploads mediante un enlace simbólico que lo que hará será establecer un puntero a la carpeta nas de cada servidor.

A continuación, iniciamos el servidor QUIZ en s1, s2 y s3, incluyendo en s1 las siguientes líneas de comandos:

npm run-script migrate_cdps

npm run-script seed_cdps

Fichero configQuiz2.sh:

```
sudo lxc-attach --clear-env -n s1 -- apt -y install nodejs
sudo lxc-attach --clear-env -n s2 -- apt -y install nodejs
sudo lxc-attach --clear-env -n s3 -- apt -y install nodejs

sudo lxc-attach --clear-env -n s1 -- apt -y install npm
sudo lxc-attach --clear-env -n s2 -- apt -y install npm
sudo lxc-attach --clear-env -n s3 -- apt -y install npm

sudo lxc-attach --clear-env -n s1 -- bash -c "
cd root;
git clone https://github.com/CORE-UPM/quiz_2019.git;
"
sudo lxc-attach --clear-env -n s2 -- bash -c "
cd root;
git clone https://github.com/CORE-UPM/quiz_2019.git;
"
sudo lxc-attach --clear-env -n s3 -- bash -c "
cd root;
git clone https://github.com/CORE-UPM/quiz_2019.git;
"
#iniciamos el servicio quiz en s1
sudo lxc-attach --clear-env -n s1 -- bash -c "
cd root;
cd quiz_2019;
cd public;
ln -s /mnt/nas uploads;
npm install;
npm install forever;
npm install mysql2;
export QUIZ_OPEN_REGISTER=yes;
export DATABASE_URL=mysql://quiz:xxxx@20.2.4.31:3306/quiz;
cd ..;
npm run-script migrate_cdps # solo en uno de los servidores;
npm run-script seed_cdps # solo en uno de los servidores;
./node_modules/forever/bin/forever start ./bin/www;
"
```

```
#iniciamos el servicio quiz en s2

sudo lxc-attach --clear-env -n s2 -- bash -c "
cd root;
cd quiz_2019;
cd public;
ln -s /mnt/nas uploads;
npm install;
npm install forever;
npm install mysql2;
export QUIZ_OPEN_REGISTER=yes;
export DATABASE_URL=mysql://quiz:xxxx@20.2.4.31:3306/quiz;
cd ..;
./node_modules/forever/bin/forever start ./bin/www
"
#iniciamos el servicio quiz en s3

sudo lxc-attach --clear-env -n s3 -- bash -c "
cd root;
cd quiz_2019;
cd public;
ln -s /mnt/nas uploads;
npm install;
npm install forever;
npm install mysql2;
export QUIZ_OPEN_REGISTER=yes;
export DATABASE_URL=mysql://quiz:xxxx@20.2.4.31:3306/quiz;
cd ..;
./node_modules/forever/bin/forever start ./bin/www
"

```

Configuración del balanceador de trafico

Para esta parte seguiremos los pasos explicados en la práctica 3 donde mediante un script escrito en Python modificaremos la configuración del tipo de balanceador empleado, el Haproxy.

Pondremos al balanceador a escuchar el puerto 80 definiendo tres servidores activos y arrancando un servidor web para la gestión del tráfico al puerto 3000 de los servidores front-end.

Fichero Python de modificación del fichero haproxy.cfg:

```python
from subprocess import call
import sys
import os
call("touch /etc/haproxy/haproxy.cfg.tmp", shell = True)
call("sudo chmod 777 /etc/haproxy/haproxy.cfg.tmp", shell = True)
f=open('/etc/haproxy/haproxy.cfg')
n=open('/etc/haproxy/haproxy.cfg.tmp', 'w')

for line in f:
    fields = line.strip().split()
    if len(fields) > 0 :
        if fields[0] == 'errorfile':
            if fields[1] == "504":
        n.write(line)
                n.write("frontend lb\nbind *:80\nmode http\ndefault_backend webservers\nbackend webservers\nmode http\nbalance roundrobin\nserver s1 20.2.3.11:3000
                continue
    n.write(line)
f.close()
n.close()

os.remove("/etc/haproxy/haproxy.cfg")
os.rename("/etc/haproxy/haproxy.cfg.tmp", "/etc/haproxy/haproxy.cfg")

call("sudo service haproxy restart ",shell=True)
```

Faltando esto en la parte derecha del fichero:

```
check\nserver s2 20.2.3.12:3000 check\nserver s3 20.2.3.13:3000 check\n")
```

Script de configuración de haproxy en el balanceador. Incluye el envío mediante el comando scp del fichero Python al balanceador, lo cual implica una autenticación para el envío.

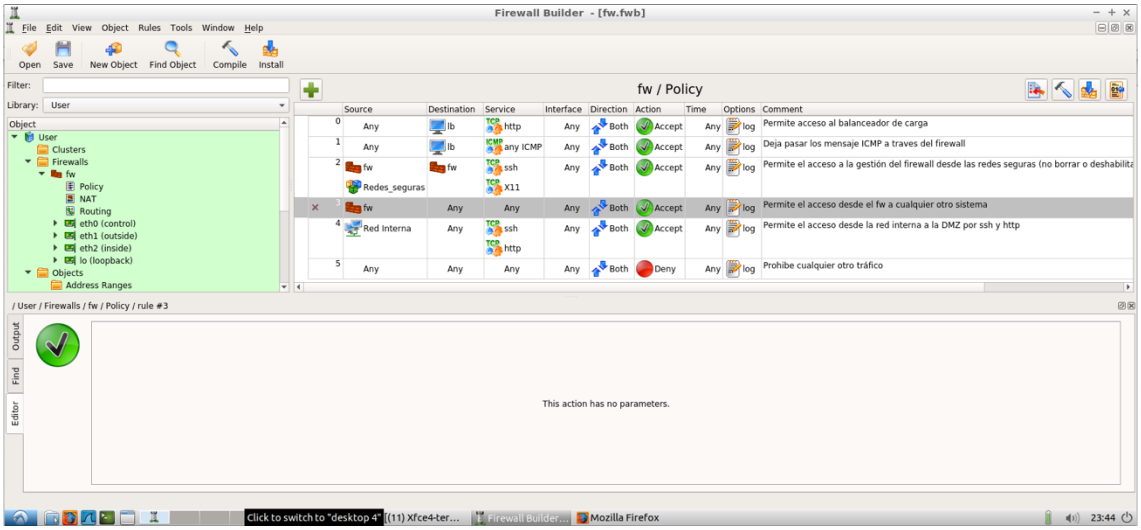
```
1  sudo lxc-attach --clear-env -n lb -- sudo apt-get update
2  sudo lxc-attach --clear-env -n lb -- sudo apt-get install haproxy
3  sudo scp confHaproxy.py root@lb:/root
4  sudo lxc-attach --clear-env -n lb -- sudo python3 root/confHaproxy.py
```

Configuración del firewall

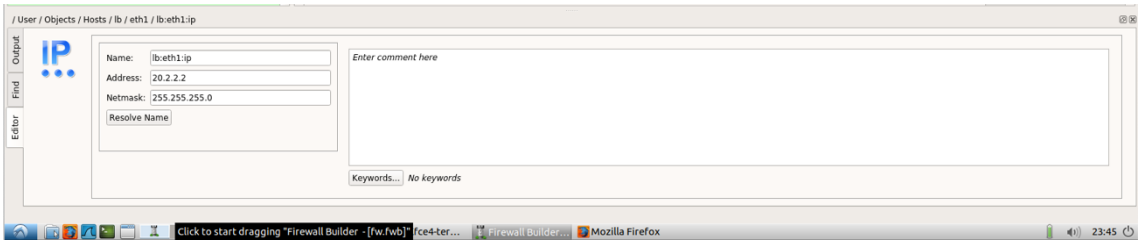
A la hora de configurar el firewall debemos cumplir con las exigencias de seguridad establecidas en el enunciado:

El firewall debe permitir únicamente el acceso mediante ping y al puerto 80 de TCP de la dirección balanceador de tráfico. Cualquier otro tráfico debe estar prohibido.

Esta directriz implica que solo se debe permitir el acceso a la dirección del balanceador de tráfico que se encuentra en la red con la que se conecta el propio firewall, LAN2. La dirección IP en esta red es 20.2.2.2. Además, debe poderse hacer peticiones de servicios web y peticiones ICMP por lo tanto el resto de servicios deben estar cubiertos. Para ello, partiendo del archivo de configuración predefinido fw.fwb establecemos las normas que permiten la entrada de tráfico ICMP y http al balanceador. En la captura aparece una norma que permite el acceso desde redes seguras al firewall, cosa que no debería ser así por cuestiones de seguridad. Esta norma debería rechazar cualquier tráfico dirigido a ambas direcciones ip del firewall, no obstante, para el desarrollo de la práctica es la única forma de acceder a este dispositivo, dado que se trata de un firewall virtualizado y conviene por tanto mantener esta norma durante la configuración.



La configuración del balanceador como objeto dentro de la interfaz gráfica:



Una vez tenemos esta configuración, que es la solicitada, descargamos el archivo ejecutable fw.fw por medio de un scp a la máquina host y escribimos el script de configuración que envía el fichero al firewall y lo ejecuta:

```
sudo scp fw.fw root@fw:/root
sudo lxc-attach --clear-env -n fw -- sudo ./root/fw.fw
```

Script de configuración completa del escenario

Una vez hemos conseguido automatizar la configuración del escenario en un orden apropiado mediante scripts que configuran cada servicio por separado, escribimos un script final que ejecute los scripts de configuración en orden.

```
sudo ./configBBDD.sh
sudo ./configGluster.sh
sudo ./configQuiz2.sh
sudo ./configLB.sh
sudo ./configFW.sh
```

Mejoras:

Script de configuración de un nuevo servidor front end

Para la configuración de un nuevo servidor front end debemos llevar a cabo los ajustes de configuración pertinentes para montar el cluster en su carpeta /mnt/nas, montar el servicio Quiz

con su correspondiente conexión a la base de datos y por último su inclusión en la configuración del balanceador de carga. Todo ello queda recogido en el script de configuración siguiente:

```
#Levantamos el servidor 4
sudo vnx -f s4.xml --create
#Montamos el nas en el servidor 4
sudo lxc-attach --clear-env -n s4 -- mkdir /mnt/nas
sudo lxc-attach --clear-env -n s4 -- mount -t glusterfs 20.2.4.21:/nas /mnt/nas
#Montamos el servicio de Quiz en el servidor 4
sudo lxc-attach --clear-env -n s4 -- apt -y install nodejs
sudo lxc-attach --clear-env -n s4 -- apt -y install npm
sudo lxc-attach --clear-env -n s4 -- bash -c "
cd root;
git clone https://github.com/CORE-UPM/quiz_2019.git;
"
sudo lxc-attach --clear-env -n s4 -- bash -c "
cd root;
cd quiz_2019;
cd public;
ln -s /mnt/nas uploads;
npm install;
npm install forever;
npm install mysql2;
export QUIZ_OPEN_REGISTER=yes;
export DATABASE_URL=mysql://quiz:xxxx@20.2.4.31:3306/quiz;
cd ..;
./node_modules/forever/bin/forever start ./bin/www
"
#Añadimos el servidor 4 al balanceador
sudo scp nuevoServHaproxy.py root@lb:/root
sudo lxc-attach --clear-env -n lb -- sudo python3 root/nuevoServHaproxy.py
```

El script nuevoServHaproxy.py contiene una modificación del fichero haproxy.cfg en la cual se añade el servidor 4:

```
from subprocess import call
import sys
import os
call("touch /etc/haproxy/haproxy.cfg.tmp", shell = True)
call("sudo chmod 777 /etc/haproxy/haproxy.cfg.tmp", shell = True)
f=open('/etc/haproxy/haproxy.cfg')
n=open('/etc/haproxy/haproxy.cfg.tmp', 'w')

for line in f:
    fields = line.strip().split()
    if len(fields) > 0 :
        if fields[0] == 'server':
            if fields[1] == "s3":
                n.write(line)
                n.write("server s4 20.2.3.14:3000 check\n")
                continue
    n.write(line)
f.close()
n.close()

os.remove("/etc/haproxy/haproxy.cfg")
os.rename("/etc/haproxy/haproxy.cfg.tmp", "/etc/haproxy/haproxy.cfg")

call("sudo service haproxy restart ",shell=True)
```

Aquí podemos ver como añadimos la dirección IP del servidor justo después de la del servidor 3.

Mejoras en el algoritmo de balanceo de carga

Para seguir criterios en base a la carga, la disponibilidad, etc. de los servidores se pueden emplear mecanismos de balanceo diferentes al round-robin sin pesos. En haproxy tenemos la posibilidad de elegir el mecanismo static round-robin con pesos añadidos de tal forma que aquel servidor que más peso tenga será el que reciba más peticiones y viceversa. También tenemos el mecanismo least connections (leastconn) que enviará la petición al servidor que menos peticiones haya recibido.

Para elegir el mecanismo más adecuado tenemos que tener en cuenta varios factores. Primero, el mecanismo static round robin resulta más útil cuando tenemos un servidor en desventaja de hardware, teniendo sentido entonces la asignación de un peso menor que otros con mejor hardware para evitar la sobrecarga de este servidor desaventajado. En nuestro caso los 3 servidores son idénticos por lo que este mecanismo no es el más adecuado.

Por otra parte, las conexiones en un escenario de uso real del servicio serían conexiones duraderas, de aproximadamente más de 10 minutos de duración, y en estos casos, la documentación del balanceador haproxy recomienda el uso de leastconn, dado que se va a basar en las conexiones que haya establecidas para realizar el reparto.

Por lo tanto, sustituiría el mecanismo round robin por el leastconn quedando el archivo de configuración de la siguiente forma:

```
balance leastconn\nserver s1 20.2.3.11:3000 check\nserver s2 20.2.3.12:3000 check\nserver s3 20.2.3.13:3000 check\n")
```

manteniéndose lo demás.