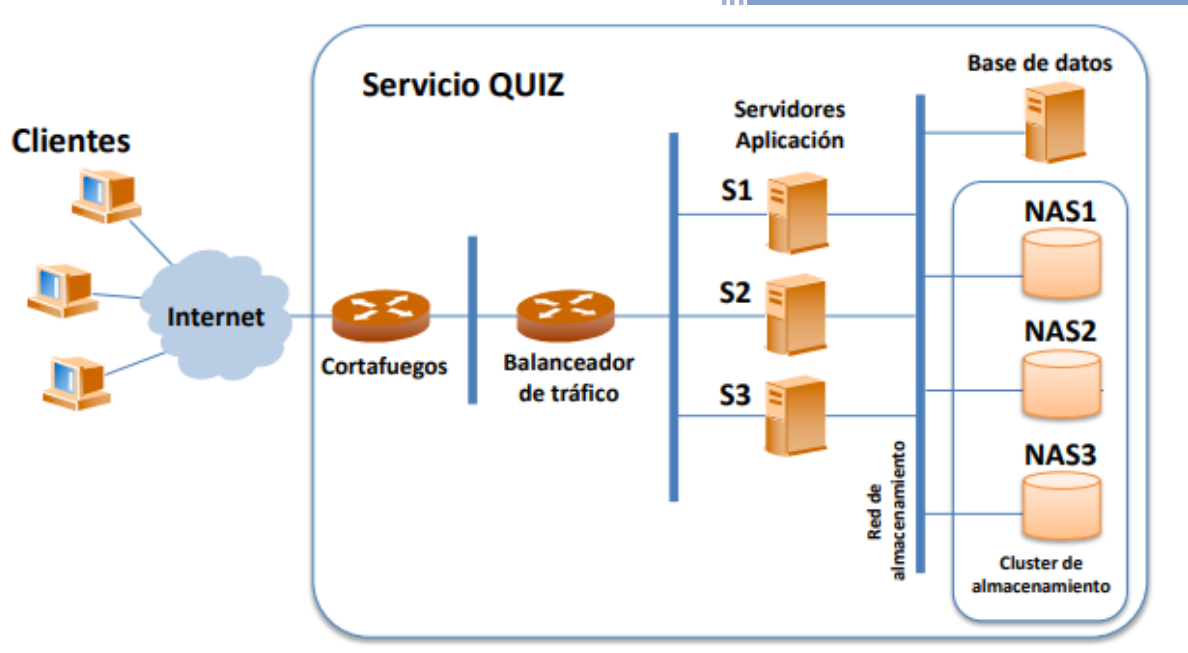


PRÁCTICA CREATIVA 2: G03



Javier de la Puente Moragón
Álvaro de Rojas Maraver
Rubén Triquero García

1. Introducción a la práctica.

Los objetivos de esta práctica, son el despliegue de una aplicación web, haciendo uso de diversas tecnologías para desplegar servicios web: firewall, balanceador de carga, tres servidores front-end corriendo la aplicación, bases de datos y servidores de almacenamiento.

Esta aplicación consistirá en un juego de quizzes, en el que el usuario ha de responder diversas preguntas que se le plantean, obteniendo una puntuación. Esta aplicación esta configurada para usar una base de datos MariaDB y las imágenes las almacenaremos en el clúster de almacenamiento el cual esta creado utilizando el fichero de almacenamiento Glusterfs. La arquitectura de esta aplicación esta sostenida también por tres servidores front-end, cuya carga será distribuida a través del balanceador de tráfico. Para añadir seguridad al sistema, el tráfico será filtrado por un firewall.

2. Configuración y arranque del escenario virtual.

Para arrancar el escenario de la práctica, hemos utilizado las instrucciones de comandos referentes a la opción de realizar la práctica desde un ordenador del laboratorio:

- Descargamos el escenario
- Descomprimos el escenario descargado.
- Instalamos los archivos necesarios.

Fichero → InstalarEscenario.py

```
#!/usr/bin/python3
import time
import os
from subprocess import call
call("cd /mnt/tmp", shell=True)
call("wget http://idefix.dit.upm.es/cdps/pc2/pc2.tgz", shell=True)
call("sudo vnx --unpack pc2.tgz", shell=True)
time.sleep(25)
os.chdir("pc2")
call("pwd")
call("bin/prepare-pc2-labo", shell=True)
time.sleep(60)
call("sudo vnx -f pc2.xml --create", shell=True)
```

Se han de añadir unos temporizadores, para asegurar que se descomprimen y se instalan los archivos correctamente, antes de continuar con la ejecución de los siguientes comandos. Esto se debe a que los comandos de descomprimir e instalar realizan funciones en segundo plano, devolviendo el control al siguiente comando, antes de haber terminado su ejecución.

Tras la ejecución del script anterior *InstalarEscenario*, podemos observar que se ha instalado correctamente:

```
Scenario "cdps_pc2_2020" started
```

VM_NAME	TYPE	CONSOLE ACCESS COMMAND
c1	lxc	con0: 'lxc-console -n c1' con1: 'lxc-console -n c1'
c2	lxc	con0: 'lxc-console -n c2' con1: 'lxc-console -n c2'
fw	lxc	con0: 'lxc-console -n fw' con1: 'lxc-console -n fw'
lb	lxc	con0: 'lxc-console -n lb' con1: 'lxc-console -n lb'
s1	lxc	con0: 'lxc-console -n s1' con1: 'lxc-console -n s1'
s2	lxc	con0: 'lxc-console -n s2' con1: 'lxc-console -n s2'
s3	lxc	con0: 'lxc-console -n s3' con1: 'lxc-console -n s3'
bbdd	lxc	con0: 'lxc-console -n bbdd' con1: 'lxc-console -n bbdd'
nas1	lxc	con0: 'lxc-console -n nas1' con1: 'lxc-console -n nas1'
nas2	lxc	con0: 'lxc-console -n nas2' con1: 'lxc-console -n nas2'
nas3	lxc	con0: 'lxc-console -n nas3' con1: 'lxc-console -n nas3'

```
Total time elapsed: 38 seconds
```

En caso de ser necesario, para arrancar el escenario virtual de la práctica, ejecutamos el archivo **ArrancarEscenario.py**:

```
#!/usr/bin/python
from subprocess import call
call("sudo vnx -f pc2/pc2.xml --start", shell=True)
```

Este script solamente es necesario ejecutarlo, después de haber parado las máquinas con shutdown, para volver a arrancarlas.

Por esta razón hemos creado un script **PararEscenario.py**, para poder parar el escenario conservando los cambios realizados en las máquinas virtuales.

```
#!/usr/bin/python
from subprocess import call
call("sudo vnx -f pc2/pc2.xml --shutdown", shell=True)
```

Por último, utilizamos el script **DestruirEscenario.py**, para parar el escenario borrando todos los cambios realizados en las máquinas virtuales.

```
#!/usr/bin/python
from subprocess import call
call("sudo vnx -f pc2/pc2.xml --destroy ", shell=True)
```

3. Configuración de servicios.

3.1 Configuración del firewall.

Para la correcta configuración del firewall, usamos la aplicación fwbuilder, que permite definir las reglas de una forma gráfica y después compilarlas a un script que se puede instalar para que se arranque cuando se arranca la máquina FW.

En la entrega, además de incluir el script de configuración del cortafuegos, incluimos el fichero fw.fw el cual es un shellscript que ejecuta todos los comandos necesarios para definir todas las reglas. Mediante ellas solo se permite el tráfico por el puerto 80 en el balanceador de tráfico, se bloquea el resto del tráfico hacia los servidores.

```
#!/usr/bin/python
from subprocess import call
#Ejecutamos fw.fw
call("sudo /lab/cdps/bin/cp2lxc /mnt/tmp/fw.fw /var/lib/lxc/fw/rootfs", shell=True)
call("sudo lxc-attach --clear-env -n fw -- chmod 777 /fw.fw", shell=True)
call("sudo lxc-attach --clear-env -n fw -- /fw.fw", shell=True)
```

Es necesario copiar el fichero fw.fw a la máquina virtual para que se ejecute allí. Como lo hemos hecho en el laboratorio, hay que copiarlo con la siguiente instrucción: `/lab/cdps/bin/cp2lxc`

A continuación, se muestra el resultado tras ejecutar el script **InstalarFirewall.py** en el terminal:

```
alvaro.derojas.maraver@l108:/$ cd /mnt/tmp
alvaro.derojas.maraver@l108:/mnt/tmp$ python3 InstalaFW.py
FILE=/mnt/tmp/fw.fw
DIR=/var/lib/lxc/fw/rootfs
Activating firewall script generated Sun Jan 10 18:56:24 2021 by root
Running prolog script
find: '/lib/modules/4.15.0-66-generic/kernel/net/': No such file or directory
find: '/lib/modules/4.15.0-66-generic/kernel/net/': No such file or directory
Verifying interfaces: eth0 eth1 eth2 lo
Rule 0 (NAT)
Rule 0 (global)
Rule 1 (global)
Rule 2 (global)
Rule 3 (global)
Rule 4 (global)
Rule 5 (global)
Rule 6 (global)
Running epilog script
```

3.2 Configuración de la base de datos.

La base de datos a utilizar será MariaDB que correrá en el servidor BBDD y que será accedida en remoto desde los servidores s1-3 para almacenar la información de la aplicación QUIZ. A continuación, creamos el script **ConfigurarBaseDatos.py** para instalar la base de datos en el servidor BBDD.

```
#!/usr/bin/python3
from subprocess import call
call("sudo lxc-attach --clear-env -n bbdd -- apt update", shell=True)
call("sudo lxc-attach --clear-env -n bbdd -- apt -y install mariadb-server", shell=True)
call("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", shell=True)
call("sudo lxc-attach --clear-env -n bbdd -- systemctl restart mysql", shell=True)
call("sudo lxc-attach --clear-env -n bbdd -- mysqladmin -u root password xxxx", shell=True)
call("sudo lxc-attach --clear-env -n bbdd -- mysql -u root --password='xxxx' -e \"CREATE USER 'quiz' IDENTIFIED BY 'xxxx';\"", shell=True)
call("sudo lxc-attach --clear-env -n bbdd -- mysql -u root --password='xxxx' -e \"CREATE DATABASE quiz;\"", shell=True)
call("sudo lxc-attach --clear-env -n bbdd -- mysql -u root --password='xxxx' -e \"GRANT ALL PRIVILEGES ON quiz.* to 'quiz'@'localhost' IDENTIFIED by 'xxxx';\"", shell=True)
call("sudo lxc-attach --clear-env -n bbdd -- mysql -u root --password='xxxx' -e \"GRANT ALL PRIVILEGES ON quiz.* to 'quiz'@'%' IDENTIFIED by 'xxxx';\"", shell=True)
call("sudo lxc-attach --clear-env -n bbdd -- mysql -u root --password='xxxx' -e \"FLUSH PRIVILEGES;\"", shell=True)
```

Hemos accedido a la base de datos desde un servidor y aquí se muestran las bases de datos que se han creado:

```
MariaDB [quiz]> show databases;
+-----+
| Database |
+-----+
| information_schema |
| quiz |
+-----+
2 rows in set (0.00 sec)
```

3.3 Configuración de GlusterFS.

Se crea un cluster de servidores de disco replicados. Se utilizará simplemente el directorio /nas del sistema de ficheros de la máquina virtual, que se exportará y sincronizará con el resto de servidores.

```
#!/usr/bin/python
from subprocess import call

#Configuracion NAS
call("sudo lxc-attach --clear-env -n nas1 -- gluster peer probe 20.20.4.22", shell=True)
call("sudo lxc-attach --clear-env -n nas1 -- gluster peer probe 20.20.4.23", shell=True)
call("sudo lxc-attach --clear-env -n nas1 -- gluster peer status", shell=True)
call("sudo lxc-attach --clear-env -n nas1 -
- gluster volume create nas replica 3 nas1:/nas nas2:/nas nas3:/nas force", shell=True)
call("sudo lxc-attach --clear-env -n nas1 -- gluster volume start nas", shell=True)
call("sudo lxc-attach --clear-env -n nas1 -- gluster volume info", shell=True)
call("sudo lxc-attach --clear-env -n nas1 -- gluster volume set nas network.ping-
timeout 5", shell=True)
call("sudo lxc-attach --clear-env -n nas2 -- gluster volume set nas network.ping-
timeout 5", shell=True)
call("sudo lxc-attach --clear-env -n nas3 -- gluster volume set nas network.ping-
timeout 5", shell=True)

#Configuracion S1-S3
call("sudo lxc-attach --clear-env -n s1 -- mkdir /mnt/nas", shell=True)
call("sudo lxc-attach --clear-env -n s1 -- mount -
t glusterfs 20.20.4.21:/nas /mnt/nas", shell=True)
call("sudo lxc-attach --clear-env -n s2 -- mkdir /mnt/nas", shell=True)
call("sudo lxc-attach --clear-env -n s2 -- mount -
t glusterfs 20.20.4.21:/nas /mnt/nas", shell=True)
call("sudo lxc-attach --clear-env -n s3 -- mkdir /mnt/nas", shell=True)
call("sudo lxc-attach --clear-env -n s3 -- mount -
t glusterfs 20.20.4.21:/nas /mnt/nas", shell=True)
```

```
Volume Name: nas
Type: Replicate
Volume ID: b187a5c0-b974-4bd6-bf6d-9ed2f77bba26
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x 3 = 3
Transport-type: tcp
Bricks:
Brick1: nas1:/nas
Brick2: nas2:/nas
Brick3: nas3:/nas
Options Reconfigured:
transport.address-family: inet
nfs.disable: on
performance.client-io-threads: off
volume set: success
volume set: success
volume set: success
```

3.4 Instalación y configuración de la aplicación quiz.

La instalación y configuración de la aplicación QUIZ en los servidores s1-s3 se realizará siguiendo las siguientes instrucciones:

```
#!/usr/bin/python
from subprocess import call

#Instalacion nodejs en S1-S3
call("sudo lxc-attach --clear-env -n s1 -- apt-get install nodejs", shell=True)
call("sudo lxc-attach --clear-env -n s2 -- apt-get install nodejs", shell=True)
call("sudo lxc-attach --clear-env -n s3 -- apt-get install nodejs", shell=True)
call("sudo lxc-attach --clear-env -n s1 -- apt-get install npm", shell=True)
call("sudo lxc-attach --clear-env -n s2 -- apt-get install npm", shell=True)
call("sudo lxc-attach --clear-env -n s3 -- apt-get install npm", shell=True)

#Clonar repositorio QUIZ
call("sudo lxc-attach --clear-env -n s1 -- bash -c \"git clone https://github.com/CORE-UPM/quiz_2021.git\"", shell=True)
call("sudo lxc-attach --clear-env -n s2 -- bash -c \"git clone https://github.com/CORE-UPM/quiz_2021.git\"", shell=True)
call("sudo lxc-attach --clear-env -n s3 -- bash -c \"git clone https://github.com/CORE-UPM/quiz_2021.git\"", shell=True)
call("sudo lxc-attach --clear-env -n s1 -- bash -c \"cd /quiz_2021; mkdir public/uploads; sed -i 's|app\\.use(r|// app\\.use(r|' app.js; npm install; npm install forever; npm install mysql2; \", shell=True)
call("sudo lxc-attach --clear-env -n s2 -- bash -c \"cd /quiz_2021; mkdir public/uploads; sed -i 's|app\\.use(r|// app\\.use(r|' app.js; npm install; npm install forever; npm install mysql2; \", shell=True)
call("sudo lxc-attach --clear-env -n s3 -- bash -c \"cd /quiz_2021; mkdir public/uploads; sed -i 's|app\\.use(r|// app\\.use(r|' app.js; npm install; npm install forever; npm install mysql2; \", shell=True)
call("sudo lxc-attach --clear-env -n s1 -- bash -c \"cd /quiz_2021; export QUIZ_OPEN_REGISTER=yes; export DATABASE_URL=mysql://quiz:xxxx@20.20.4.31:3306/quiz; npm run-script migrate_env; npm run-script seed_env; ./node_modules/forever/bin/forever start ./bin/www \", shell=True)
call("sudo lxc-attach --clear-env -n s2 -- bash -c \"cd /quiz_2021; export QUIZ_OPEN_REGISTER=yes; export DATABASE_URL=mysql://quiz:xxxx@20.20.4.31:3306/quiz; ./node_modules/forever/bin/forever start ./bin/www \", shell=True)
call("sudo lxc-attach --clear-env -n s3 -- bash -c \"cd /quiz_2021; export QUIZ_OPEN_REGISTER=yes; export DATABASE_URL=mysql://quiz:xxxx@20.20.4.31:3306/quiz; ./node_modules/forever/bin/forever start ./bin/www \", shell=True)

#Enlace simbolico
call("sudo lxc-attach --clear-env -n s1 -- bash -c \"cd /quiz_2021/public; ln -s /mnt/nas uploads\"", shell=True)
call("sudo lxc-attach --clear-env -n s2 -- bash -c \"cd /quiz_2021/public; ln -s /mnt/nas uploads\"", shell=True)
call("sudo lxc-attach --clear-env -n s3 -- bash -c \"cd /quiz_2021/public; ln -s /mnt/nas uploads\"", shell=True)
```

Estas instrucciones tienen las siguientes funciones:

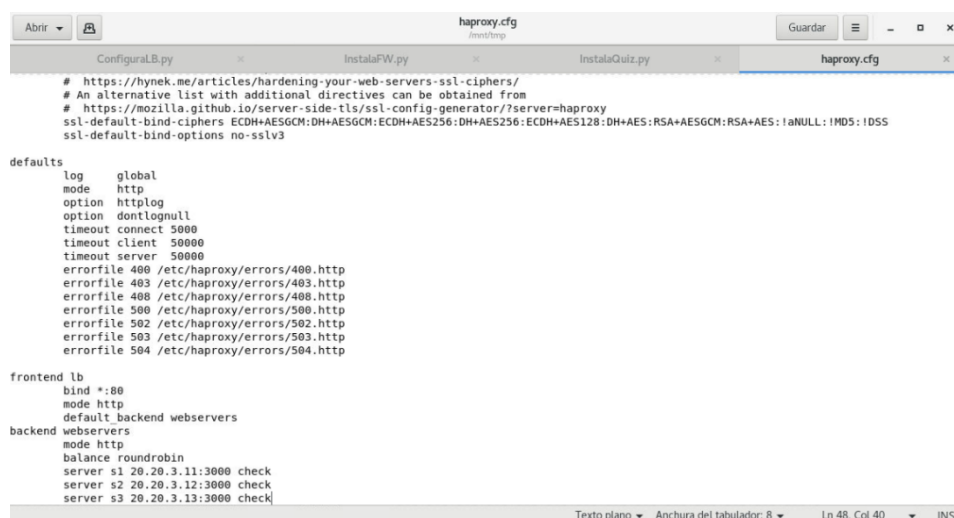
- Clonar el repositorio e instalarlo, hay que asegurarse que se cambia el puerto en el que trabaja.
- Arrancar el servidor QUIZ en background, exportando la base de datos a la máquina BBDD.
- Por último, crear el enlace simbólico que permita copiar las imágenes al clúster de almacenamiento.

3.5 Configuración del balanceador de tráfico.

```
#!/usr/bin/python
from subprocess import call
import time

#Instalamos haproxy
call("sudo lxc-attach --clear-env -n lb -- bash -c \"apt-get update; apt-
get install haproxy\"", shell=True)
call("sudo lxc-attach --clear-env -n lb -- sudo rm /etc/haproxy/haproxy.cfg", shell=True)
#Volvemos a copiar haproxy al directorio origen
call("sudo /lab/cdps/bin/cp2lxc haproxy.cfg /var/lib/lxc/lb/rootfs/etc/haproxy", shell=True)
time.sleep(5)
#Reiniciamos haproxy
call("sudo lxc-attach --clear-env -n lb -- sudo service haproxy restart", shell=True)
```

- El fichero haproxy.cfg lo incluimos en el zip del escenario. Se copia a la máquina del balanceador de tráfico y se configura dicho servicio.
- Hemos configurado el balanceador con un algoritmo round-robin, y los tres servidores como backend web servers



```
haproxy.cfg
# https://hynke.me/articles/hardening-your-web-servers-ssl-ciphers/
# An alternative list with additional directives can be obtained from
# https://mozilla.github.io/server-side-tls/ssl-config-generator/?server=haproxy
ssl-default-bind-ciphers ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:RSA+AESGCM:RSA+AES:1aNULL:1MD5:1DSS
ssl-default-bind-options no-sslv3

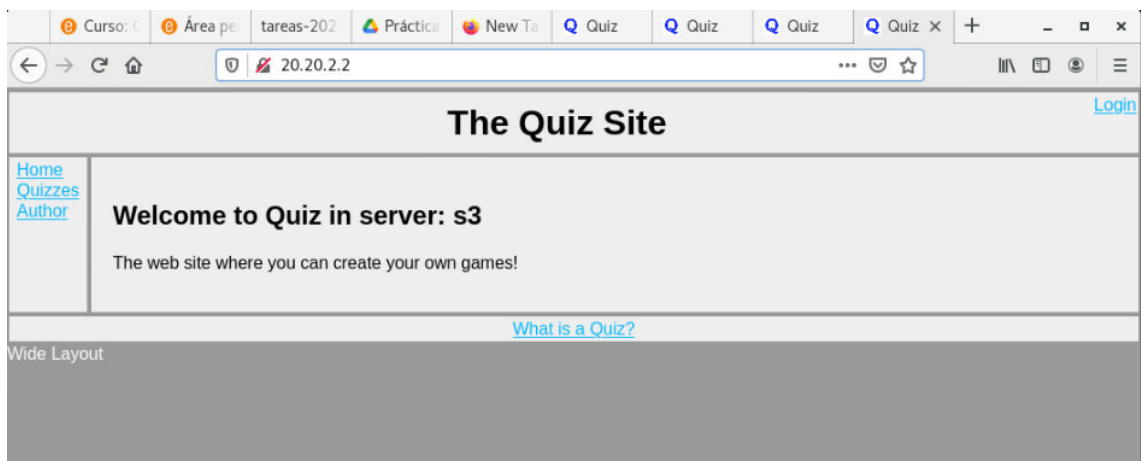
defaults
    log          global
    mode         http
    option        httplog
    option        dontlognull
    timeout connect 5000
    timeout client  50000
    timeout server  50000
    errorfile 400 /etc/haproxy/errors/400.http
    errorfile 403 /etc/haproxy/errors/403.http
    errorfile 408 /etc/haproxy/errors/408.http
    errorfile 500 /etc/haproxy/errors/500.http
    errorfile 502 /etc/haproxy/errors/502.http
    errorfile 503 /etc/haproxy/errors/503.http
    errorfile 504 /etc/haproxy/errors/504.http

frontend lb
    bind *:80
    mode http
    default_backend web servers

backend web servers
    mode http
    balance roundrobin
    server s1 20.20.3.11:3000 check
    server s2 20.20.3.12:3000 check
    server s3 20.20.3.13:3000 check
```

En la imagen anterior se muestra el fichero haproxy.cfg

Observamos que cada vez que nos conectamos al servicio, dependiendo de la carga que haya en ese momento, nos redirige a un servidor distinto.



4. Puntos débiles del escenario.

Los puntos débiles en cuanto a fiabilidad, escalabilidad, se pueden resumir en:

- La base de datos, al no estar replicada, si se cae, el servicio deja de funcionar, generando problemas de fiabilidad. Para solucionar esto recomendamos replicar la base de datos.
- En caso de haber mucha carga en la red, habría que aumentar el número de servidores. Esto genera un problema de escalabilidad.
- Podría mejorarse el escenario, cambiando el algoritmo de configuración round-robin del balanceador por otro más eficiente para este escenario concreto.