

# **PRÁCTICA CREATIVA 2:**

## **DESPLIEGUE DE UNA APLICACIÓN ESCALABLE**

### **Autores:**

- Diego García Fierro
- Yamil Mateo Rodríguez

### **Descripción**

El objetivo de la práctica consiste en la replicación de una arquitectura de despliegue de una aplicación QUIZ propuesta, que combina algunos de los elementos de arquitecturas más empleados en la actualidad con conceptos estudiados en la asignatura.

La configuración de la arquitectura propuesta se ha llevado a cabo elemento a elemento en función del orden en el que intervienen para conseguir el correcto funcionamiento de la aplicación, de forma que el diseño y las decisiones tomadas durante la implantación se han llevado a cabo de forma constructiva y coordinada entre los elementos.

Además, se han realizado varias mejoras sobre la arquitectura básica propuesta con el fin de incrementar la escalabilidad de la aplicación y optimizar su funcionamiento así como el control de los procesos que tienen lugar a nivel interno.

### **Arquitectura básica de la aplicación: decisiones de diseño e implementación**

La arquitectura de la aplicación consiste en los siguientes elementos, que se han ido implementando de forma secuencial uno a uno de forma que su comportamiento fuera coordinado:

- Cortafuegos. Encargado de filtrar e impedir el paso de todo aquel tráfico ajeno al contexto de la aplicación.
- Base de datos. Contiene la información asociada a la aplicación QUIZ.
- Cluster de almacenamiento. Almacena las imágenes que se emplean en la aplicación QUIZ. Compuesto por tres servidores nas.
- Servidores de aplicación. Albergan la lógica asociada a la aplicación QUIZ -todos los servidores por igual- y permiten la prestación del servicio a los clientes.
- Balanceador de tráfico. Su función radica en la distribución del tráfico que tiene a los servidores de aplicación como destino, para así evitar la congestión de los mismos.

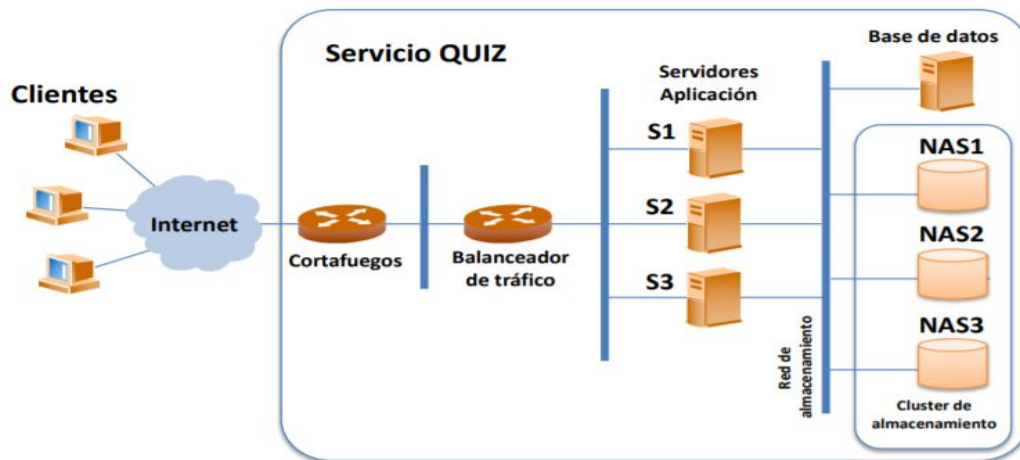


Figura 1: Arquitectura básica de la aplicación QUIZ.

Para la ejecución de muchos de los comandos en los scripts de configuración de cada elemento de la arquitectura se emplea el fichero *commands.yaml*. El principal motivo por el que se ha elegido YAML para este propósito es por las facilidades que proporciona a la hora de emplear comandos que tienden a repetirse en los scripts de configuración de los elementos.

## 1. Cortafuegos

La implementación del cortafuegos se realiza a partir de la aplicación *fwbuilder*, de manera que se define una *policy* con las reglas de comportamiento del cortafuegos en la interfaz gráfica del programa y se genera el fichero *fw.fw* listo para ser instalado en la máquina virtual que actúa como cortafuegos.

La *policy* con las reglas de comportamiento del fichero *fw.fw* guardan bastante similitud con la configuración de ejemplo del fichero *fw.fwb*. Sin embargo, como se observa en la Figura 2, en el caso de *fw.fw* no solo se permite el acceso web a *s1*, sino que se configura el acceso a cualquiera de los servidores de aplicación a todo el tráfico http, y se hace especificando el balanceador de tráfico como destino, ya que es este el encargado de redirigir el tráfico hacia los servidores de aplicación en función de la saturación de los mismos. Además, para la implementación de esta configuración es necesaria la creación de un nuevo objeto en *fwbuilder* asociado al balanceador de tráfico, que denominaremos 'lb' (Figura 3). Existen otras peculiaridades con respecto a las reglas de la *policy* creada, que se corresponden con algunas de las mejoras implementadas y se describirán más adelante en este documento.

fw / Policy								
	Source	Destination	Service	Interface	Direction	Action	Time	Options
0	Any	lb	TCP http ICMP any ICMP TCP HAProxyService	Any	Both	Accept	Any	log
1	fw Redes_seguras	fw	TCP ssh TCP X11	Any	Both	Accept	Any	log
2	fw	Any	Any	Any	Both	Accept	Any	log
3	Red Interna	Any	TCP ssh TCP http	Any	Both	Accept	Any	log
4	Any	Any	Any	Any	Both	Deny	Any	log

Figura 2: Policy con las reglas de comportamiento del cortafuegos.

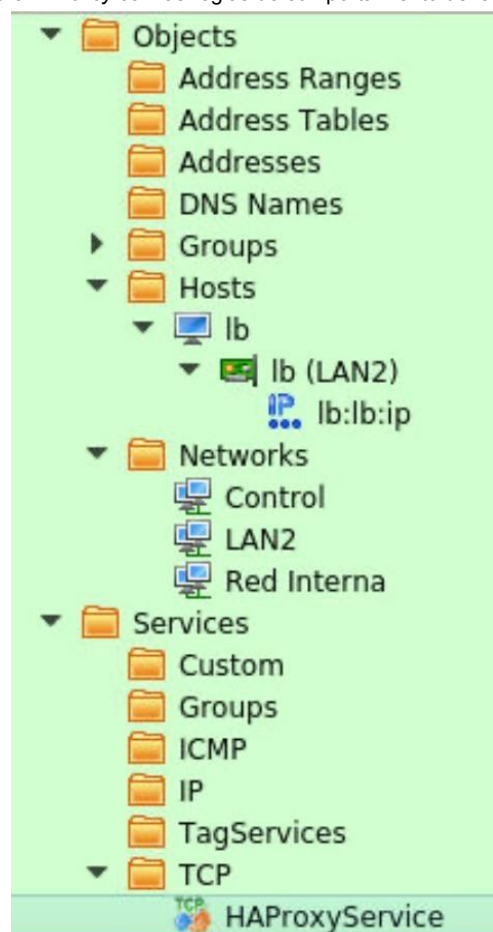


Figura 3: Objetos de la policy creada para configurar el cortafuegos.

A nivel de instalación y ejecución del fichero *fw.fw* creado se utiliza el script *firewallConf.py*. Este script se encarga de crear un directorio */etc/fw* en la máquina virtual asociada al cortafuegos ('fw'), copia el fichero *fw.fw* en dicho directorio y lo ejecuta para así proceder con la instalación de la *policy* en el cortafuegos.

## 2. Base de datos

Antes de la instalación de la aplicación QUIZ en los servidores de aplicación, se necesita primero configurar la base de datos que va a contener la información asociada a los quizzes, del mismo modo que posteriormente se necesita configurar el cluster de almacenamiento de las imágenes de la aplicación.

Como base de datos de almacenamiento de la información asociada a la aplicación QUIZ se emplea MariaDB. La base de datos se instala en la máquina virtual 'bbdd' y para ello se siguen paso a paso las indicaciones descritas en el enunciado de la práctica.

El script creado para la configuración de la base de datos es *databaseInstaller.py*. En primera instancia se intentó que los comandos a ejecutar se importaran desde *commands.yaml*, pero la gran cantidad de problemas que surgieron, especialmente con el escapado, hicieron que finalmente se recurriera a introducir los comandos para la instalación de la base de datos directamente en el script.

### 3. Cluster de almacenamiento

De la misma forma que ocurría con la base de datos, antes de proceder con la instalación de la aplicación se debe configurar el cluster de almacenamiento. Se trata de un cluster formado por tres servidores de disco (*nas1-3*), y el script empleado para su configuración es *GlusterInstaller.py*. Además, el cluster se crea a partir del sistema de ficheros distribuidos *Glusterfs*.

El script está compuesto por dos funciones: *NASconf(nNas, lc)* y *MountNas(nServ, lc)*. La primera función se encarga de la configuración de los servidores de disco del cluster, mientras que la segunda trata el montaje de los servidores nas desde los servidores de aplicación.

Para la configuración de los servidores de disco del cluster se crea una lista con los ids de los *nas*, que dependen del parámetro que se le pase a la función. Esto proporciona un gran grado de generalidad a la implementación, ya que la función permite crear clusters con el número de nas que se deseen (aunque en este caso solo sean tres). Por otro lado, la creación del volumen con tres servidores que replican la información se lleva a cabo mediante el siguiente comando:

```
call(preStr + " nas1 -- gluster volume create nas replica 3 20.20.4.2"+ str(nasIDS[0])  
+ ":/nas/ 20.20.4.2"+str(nasIDS[1])+":/nas/ 20.20.4.2"+str(nasIDS[2])+":/nas/ force", shell =  
True)
```

El parámetro '*replica*' se emplea para indicar el número de réplicas de la información que se desean crear. La estructura '*hostX : brickX*' permite especificar la IP de cada nas (host) y el directorio del sistema de ficheros de la máquina virtual que se exporta y sincroniza con el resto de servidores (brick). La opción '*force*' fuerza la creación instantánea del volumen.

Por otro lado, la configuración del montaje desde los servidores web se lleva a cabo principalmente para garantizar el acceso al sistema de ficheros exportado por los *nas* desde los servidores de aplicación. Se emplea el comando *mount*, y de esta forma */mnt/nas* se convierte en el directorio de los servidores de aplicación en el cual se puede visualizar el contenido exportado por los *nas*.

### 4. Servidores de aplicación

Una vez se han configurado los recursos necesarios (base de datos y cluster de almacenamiento) para el correcto despliegue de la aplicación QUIZ en los servidores, se crea el script *serverQuizConf.py* con dicho objetivo.

Como ya ocurría con la configuración de la base de datos, la instalación y configuración de la aplicación QUIZ en los tres servidores de la arquitectura básica es muy intuitiva y simplemente consiste en seguir los pasos marcados por el enunciado de la práctica.

El comando *sed* recomendado para borrar la línea indicada en el fichero */quiz\_2021/app.js* se emplea con la opción *'-i '29d'* (se corresponde con la línea número 29 del fichero), y los valores asignados a la variable *DATABASE\_URL* son los siguientes:

```
DATABASE_URL=mysql://quiz:xxxx@20.20.4.31:3306/quiz
```

Las migraciones y la ejecución del *seeder*, que solo deben llevarse a cabo en uno de los servidores, se realizan en el servidor *s1*. Además, se le debe indicar a los servidores de aplicación que */public/uploads/* es el directorio de los servidores *nas* en el cual se albergan las imágenes, para lo cual se emplea la siguiente línea:

```
mount -t glusterfs 20.20.4.21:/nas public/uploads\
```

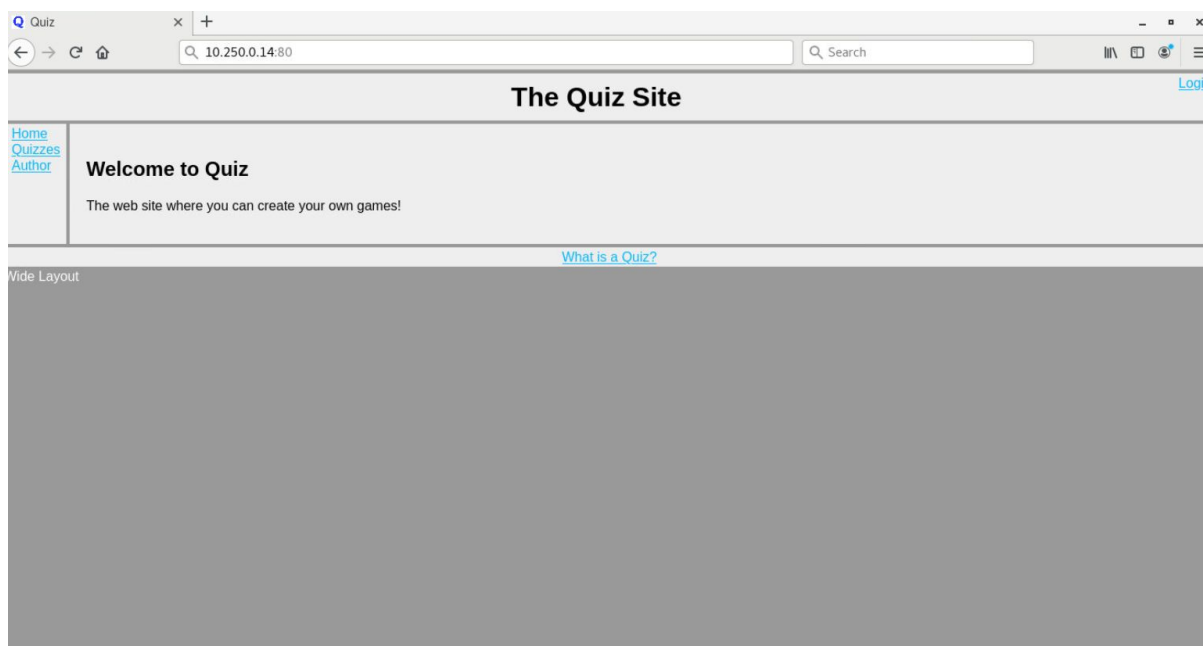


Figura 4: Aplicación QUIZ accesible en el puerto 80 de la IP del balanceador de tráfico.

## 5. Balanceador de tráfico

Por último, se lleva a cabo la configurador del balanceador de tráfico, necesario para distribuir la carga entre los distintos servidores de aplicación. Se emplea el balanceador HAproxy utilizado en la práctica creativa 1 (<http://www.haproxy.org/>), basado en un algoritmo round-robin y que se configura mediante el script *LBinstaller.py*, que además genera el fichero de configuración *haproxy.cfg*. Tras la configuración del balanceador de tráfico este ha de ser reiniciado para su correcta puesta en funcionamiento.

## Mejoras implementadas

Además de la arquitectura básica propuesta en el enunciado, se han llevado a cabo algunas de las partes opcionales recomendadas que permiten mejorar las prestaciones del escenario. Estas mejoras implementadas son:

**1. 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.**

La configuración básica del firewall incluye reglas de comportamiento que garantizan el acceso del tráfico http asociado a la aplicación y con destino en el balanceador de tráfico, de forma que este pueda distribuir dicho tráfico entre los distintos servidores de aplicación.

Además, otras de sus reglas permiten el paso del tráfico ICMP hasta el balanceador de tráfico, el acceso al firewall desde redes seguras para establecer su configuración, la comunicación del firewall con el resto de equipos que integran el escenario desplegado y el tráfico (de cualquier tipo) emitido desde la red interna del escenario y que tenga como destino cualquier otra red externa.

El resto del tráfico, por tanto, es filtrado por el firewall, que garantiza la integridad del escenario. Sin embargo, en esta mejora se quiere permitir el acceso mediante ping al puerto 80 de TCP de la dirección del balanceador de tráfico. Esto implica la creación de un nuevo objeto y una nueva regla de comportamiento en la *policy* del firewall.

El objeto *HAProxyService* ha de ser creado en la carpeta *Services*, concretamente en *TCP*, y en él se han de especificar la dirección IP del balanceador de tráfico y el puerto (80) en el que se atiende el tráfico recibido (ver Figura 3). Por su parte, y como se puede ver en la Figura 5, la regla que rige el acceso por ping a la dirección del balanceador es la misma que permite el acceso del tráfico HTTP y el tráfico ICMP a la dirección del balanceador de tráfico.

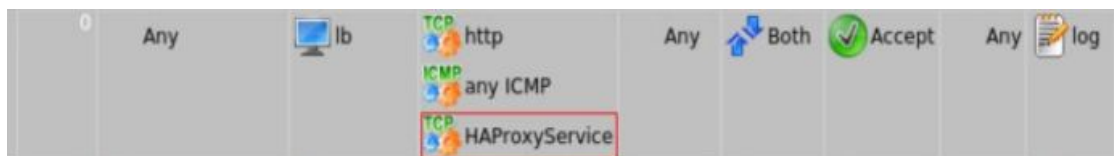


Figura 5: Regla que incluye el objeto HAProxyService.

**2. Crear el procedimiento para añadir y configurar un nuevo servidor front-end adicional. Para se puede utilizar el escenario s4.xml, que añade un nuevo servidor s4 al escenario.**

Para la realización de esta mejora se emplea el fichero *s4.xml* proporcionado. El objetivo es modificar dicho fichero para configurarlo como el resto de servidores front-end que ya están implementados en el escenario de la práctica (s1-3), y una vez se ha conseguido adaptarlo incluirlo en el fichero de configuración *pc2.xml*.

De esta forma, se crea el script *addServer.py* para dicho propósito, y en él se emplea la librería *ElementTree* que permite la edición de ficheros *.xml*. En primer lugar, la función propuesta edita el fichero *s4.xml* para que contenga las mismas propiedades que cualquiera de los servidores front-end que forman parte del fichero *pc2.xml* principal del escenario. La librería utilizada permite obtener las propiedades

'exec' y 'filetree' que tienen los servidores front-end (se copian de *s1* en *pc2.xml*) y añadirlas a *s4* en *s4.xml*.

Sin embargo, añadir nuevas propiedades al fichero *s4.xml* hace que se pierda el formato y aparezcan múltiples espaciados que impiden la correcta interpretación del fichero. Por ello, se debe llevar una segunda adaptación del fichero que permita conseguir el formato apropiado antes de copiar su contenido en *pc2.xml*.

Una vez se ha conseguido la adaptación de *s4.xml* se debe copiar su contenido en el fichero de configuración principal del escenario. Tras conseguir la interpretación del fichero mediante un parser, la siguiente línea permite incluir la configuración xml de *s4* justo a la altura de los servidores front-end en *pc2.xml*:

```
globalFile.getroot().insert(13, baseServerFile.getroot())
```

En última instancia se utiliza la función *unescape* con el fichero *pc2.xml* para evitar problemas con el escapado. De esta forma, al proceder con la ejecución del escenario mediante el comando *python3 pc2.py* la máquina virtual *s4* se inicializará como un servidor front-end más en la arquitectura propuesta.

### **3. Utilizar un sistema de replicación de bases de datos para añadir resistencia a fallos.**

Para realizar esta mejora, se ha creado una nueva máquina virtual en la que se va a replicar la base de datos principal. No va a ser una simple clonación de la base de datos, si no que se va a establecer un sistema de 'master-slave' siendo master la máquina virtual, '*bbdd*', de la que se replican los eventos y slave la máquina virtual, '*sbbdd*', que es la que replica a master. Para conseguir este sistema, se crea un usuario con permisos de esclavo en master, en slave se indica a que host se tiene que conectar y que usuario usar para establecer el rol de esclavo.

Esta mejora también ha llevado a modificaciones en los archivos de configuración de las bases de datos y cambiar el orden en el que crear la base de datos 'quiz', que ahora se crea tras haber establecido el sistema.

A continuación se muestra su correcto funcionamiento:

```

MariaDB [(none)]> show slave status \G
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
        Master_Host: 20.20.4.31
        Master_User: replication
        Master_Port: 3306
        Connect_Retry: 60
        Master_Log_File: mysql-bin.000001
        Read_Master_Log_Pos: 14555
        Relay_Log_File: mysql-relay-bin.000002
        Relay_Log_Pos: 14843
        Relay_Master_Log_File: mysql-bin.000001
        Slave_IO_Running: Yes
        Slave_SQL_Running: Yes
        Replicate_Do_DB:
        Replicate_Ignore_DB:
        Replicate_Do_Table:
        Replicate_Ignore_Table:
        Replicate_Wild_Do_Table:
        Replicate_Wild_Ignore_Table:
          Last_Errno: 0
          Last_Error:
          Skip_Counter: 0
        Exec_Master_Log_Pos: 14555
        Relay_Log_Space: 15141
        Until_Condition: None

```

```

sbbdd - console #1
      Replicate_Wild_Do_Table:
      Replicate_Wild_Ignore_Table:
        Last_Errno: 0
        Last_Error:
        Skip_Counter: 0
        Exec_Master_Log_Pos: 14555
        Relay_Log_Space: 15141
        Until_Condition: None
        Until_Log_File:
        Until_Log_Pos: 0
        Master_SSL_Allowed: No
        Master_SSL_CA_File:
        Master_SSL_CA_Path:
        Master_SSL_Cert:
        Master_SSL_Cipher:
        Master_SSL_Key:
      Seconds_Behind_Master: 0
      Master_SSL_Verify_Server_Cert: No
        Last_IO_Errno: 0
        Last_IO_Error:
        Last_SQL_Errno: 0
        Last_SQL_Error:
      Replicate_Ignore_Server_Ids:
        Master_Server_Id: 1
        Master_SSL_Crl:
        Master_SSL_Crlpath:
          Using_Gtid: No
          Gtid_IO_Pos:
      Replicate_Do_Domain_Ids:
      Replicate_Ignore_Domain_Ids:
        Parallel_Mode: conservative
1 row in set (0.00 sec)

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| quiz |
+-----+
4 rows in set (0.00 sec)

MariaDB [(none)]>

```

```

bbdd - console #1
* Management:      https://landscape.canonical.com
* Support:         https://ubuntu.com/advantage
root@bbdd:~# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with
Your MariaDB connection id is 51
Server version: 10.1.47-MariaDB-0ubuntu0.18.04.1 U

Copyright (c) 2000, 2018, Oracle, MariaDB Corporati

Type 'help;' or '\h' for help. Type '\c' to clear t

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| quiz |
+-----+
4 rows in set (0.00 sec)

MariaDB [(none)]>

```

4. Realizar mejoras en el algoritmo de balanceo de carga para seguir criterios de asignación en base a la carga, disponibilidad, etc. de los servidores.



En cuanto a las mejoras del algoritmo de balanceo, se ha decidido establecer un round-robin con pesos. Otra opción, era hacer que se conectarán según el menor número de conexiones, con el algoritmo 'leastconn' de HAProxy, esto carecía de sentido con la aplicación quiz porque es recomendable para aplicaciones con sesiones largas de comunicación entre cliente y servidor, con esta aplicación las sesiones eran muy cortas. Por lo tanto, se ha emulado un leastconn ajustando los pesos del round robin, por lo tanto, teniendo 4 servidores lo que hemos establecido son 35, 25, 20, 20 respectivamente para cada servidor.

En la siguiente imagen se puede ver que funciona:

## HAProxy

### Statistics Report for pid 876

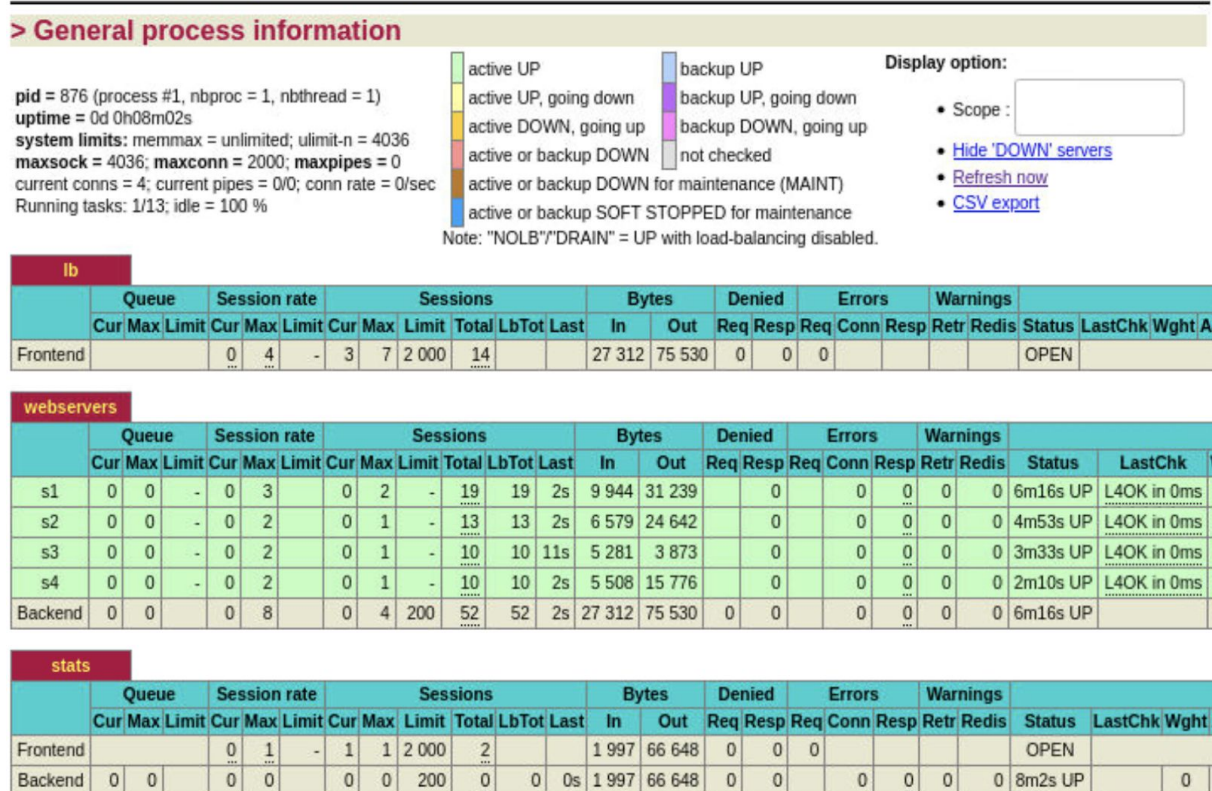


Figura 6: Demostración estadísticas HAProxyService.

#### 5. Consolidación de los logs de todos los servidores en un servidor centralizado (una nueva máquina virtual llamada logs) mediante, por ejemplo, rsyslog o logstash.

En muchas ocasiones, los administradores de sistemas pasan mucho tiempo para localizar los logs que proporcionan información relevante sobre eventos y procesos ejecutados en dichos sistemas. Si el escenario está formado por diversas máquinas, como ocurre en este caso, cada una de ellas tendrá sus propios logs almacenados de forma local. Por este motivo, plantear un sistema de logging centralizado en el escenario garantiza una mayor comodidad y seguridad a la hora de acceder a los ficheros de logs.

Para la realización de esta mejora se parte de la propuesta de la creación de un nuevo servidor centralizado (nueva máquina virtual llamada 'logs') que reúna los logs

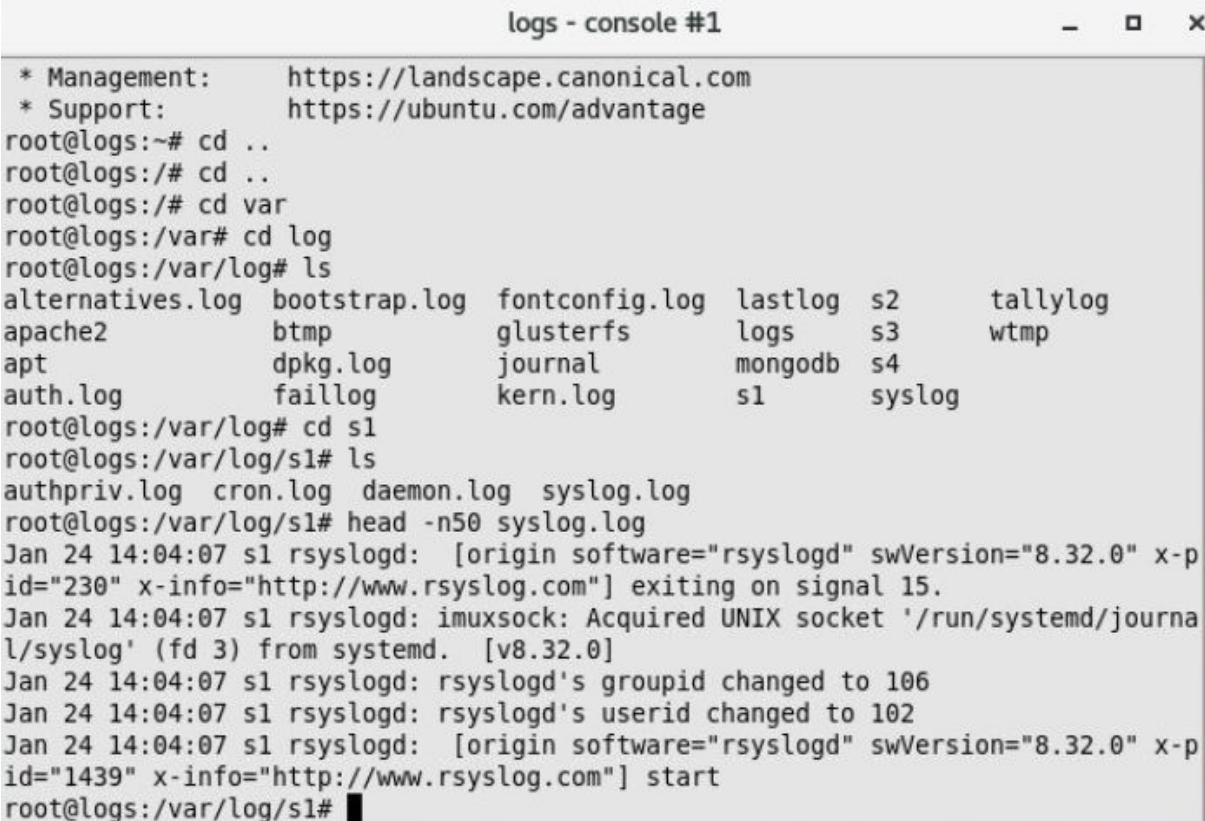
emitidos en cada uno de los servidores de aplicación (s1-4). Además, se emplea *rsyslog* como herramienta para manejar los ficheros de logs de los sistemas.

Se crea el fichero *logsConf.py* con las funciones que implementan la consolidación de los logs en un servidor centralizado. La configuración con *rsyslog* se basa en definir la nueva máquina virtual *logs* como servidor rsyslog, de forma que este reciba todos los logs generados por los servidores de aplicación, que a su vez se tratan como clientes rsyslog.

En primer lugar se debe instalar y habilitar *rsyslog*, tanto en el servidor centralizado como en los servidores de aplicación. La función *rsyslogServer(cm)* crea el fichero de configuración *rsyslog* del servidor centralizado, definiendo que el servidor acepte cualquier log en su puerto 514 utilizando TCP para la transmisión. Además, se especifica que los logs de los servidores de aplicación deben almacenarse en el directorio */var/log/%HOSTNAME* (siendo HOSTNAME una propiedad de *rsyslog* que representa a la máquina de la que provienen los logs) del servidor centralizado.

Por su parte, la configuración de *rsyslog* en los servidores de aplicación se lleva a cabo mediante la función *rsyslogCliente(cm)*. De la misma forma que ocurría con el servidor centralizado, se crea un nuevo fichero de configuración en el que se da la orden de mandar todo tipo de logs generados por la máquina a la dirección IP de la máquina virtual *logs*.

De esta forma, y como se puede observar en la Figura 6, se consigue la consolidación de los logs en el servidor centralizado.



```
logs - console #1
* Management:      https://landscape.canonical.com
* Support:         https://ubuntu.com/advantage
root@logs:~# cd ..
root@logs:/# cd ..
root@logs:/# cd var
root@logs:/var# cd log
root@logs:/var/log# ls
alternatives.log  bootstrap.log  fontconfig.log  lastlog        s2            tallylog
apache2           btmp          glusterfs       logs           s3            wtmp
apt              dpkg.log      journal         mongoddb       s4
auth.log          faillog       kern.log        s1            syslog
root@logs:/var/log# cd s1
root@logs:/var/log/s1# ls
authpriv.log  cron.log  daemon.log  syslog.log
root@logs:/var/log/s1# head -n50 syslog.log
Jan 24 14:04:07 s1 rsyslogd: [origin software="rsyslogd" swVersion="8.32.0" x-p
id="230" x-info="http://www.rsyslog.com"] exiting on signal 15.
Jan 24 14:04:07 s1 rsyslogd: imuxsock: Acquired UNIX socket '/run/systemd/journa
l/syslog' (fd 3) from systemd. [v8.32.0]
Jan 24 14:04:07 s1 rsyslogd: rsyslogd's groupid changed to 106
Jan 24 14:04:07 s1 rsyslogd: rsyslogd's userid changed to 102
Jan 24 14:04:07 s1 rsyslogd: [origin software="rsyslogd" swVersion="8.32.0" x-p
id="1439" x-info="http://www.rsyslog.com"] start
root@logs:/var/log/s1#
```

Figura 7: Logs en el servidor centralizado 'logs'.

6. Indicar cómo cambiaría el despliegue en caso de desplegarlo en OpenStack, Amazon AWS o Google Cloud.

En esta mejora se van a describir las principales líneas de actuación para el despliegue del escenario en la plataforma de Amazon AWS, aunque el planteamiento sería similar para el caso de Google Cloud.

Dado que todos los elementos empleados en el escenario original son máquinas virtuales ligeras basadas en *LXC*, bastaría con desplegar instancias de máquinas virtuales en AWS de manera que la configuración de cada una de ellas (firewall, balanceador de tráfico, servidores de aplicación, base de datos y cluster de almacenamiento) se siguiera llevando a cabo mediante ficheros *.py*, como se había planteado inicialmente en esta práctica. Sin embargo, en este caso de despliegue cada fichero debería ejecutarse desde la consola *AWS CodeDeploy* que AWS proporciona para manejar las máquinas virtuales.

Además, algunas de las máquinas virtuales provistas cuentan con el software necesario para su configuración. En el caso del firewall, la *policy* que rige su comportamiento se crea mediante la interfaz gráfica de la aplicación *fwbuilder*. La base de datos empleada es *MariaDB*, mientras que el cluster de almacenamiento se configura a partir del sistema de ficheros *GlusterFS*. En cualquiera de los casos mencionados, se deberán instalar las dependencias y los paquetes necesarios en las máquinas virtuales de AWS para que los ficheros *.py* puedan ejercer su función correctamente.

AWS proporciona guías muy detalladas para el despliegue de aplicaciones web que se extraen desde *GitHub*. En este caso se parte de las 4 instancias de máquinas virtuales AWS ya creadas que simulan cada uno de los 4 servidores de aplicación implementados en el diseño del escenario. Para el despliegue de la aplicación desde la interfaz gráfica de AWS en cada uno de los servidores se han de seguir los siguientes pasos:

1. Creación de la aplicación y de un grupo de despliegue. Han de crearse para poder desplegar la aplicación que se extraerá del repositorio de *GitHub*. En el caso de la aplicación (*Navigation > Deploy > Applications*), deberá especificarse su nombre y su *Compute Platform* (que deberá ser *EC2/On-premises*). El grupo de despliegue, que se crea desde el menú *Deployment groups*, deberá contener un nombre, un *Service role* (depende del servicio a proveer), un *Deployment type* (*In-place*) y un *Environment configuration* (o bien *Amazon EC2 instances* o *On-premises instances*). También cabe la posibilidad de crear la aplicación y el grupo de despliegue por defecto en la CLI provista por AWS.
2. Despliegue de la aplicación en la instancia. Se puede realizar desde la interfaz gráfica de AWS y desde la CLI de AWS. Consiste en configurar la conexión de AWS con la cuenta de *GitHub* que contiene el repositorio con el código de la aplicación. Una vez se han sincronizado ambas cuentas, se asocia el repositorio de *GitHub* con la aplicación y el grupo de despliegue creados anteriormente. La propia interfaz gráfica de AWS guía de forma muy detallada en este proceso.
3. Verificación del despliegue. Si la aplicación y el grupo de despliegue se han sincronizado correctamente con el repositorio de *GitHub*, aparecerá 'Succeeded' en la columna de estado del menú disponible en *Navigation > Deploy > Deployments*. Además, se puede comprobar el correcto funcionamiento de la aplicación en el navegador introduciendo <http://public-dns> (según la instancia).

La documentación oficial de AWS provee una guía mucho más completa sobre el despliegue de aplicaciones web en instancias de máquinas virtuales.

De esta forma, y tras haber ejecutado los correspondientes ficheros de configuración, se habrá procedido con el despliegue del escenario propuesto utilizando Amazon AWS.

## **Discusión final: problemas detectados y posibles soluciones**

El despliegue de la aplicación propuesta, a pesar de poder ser considerado eficiente con los recursos disponibles, presenta algunos problemas que se han detectado y sus posibles soluciones podrían mejorar las prestaciones del escenario.

Por un lado, dado que el despliegue del escenario se lleva a cabo a través de la ejecución del fichero *pc2.py*, que contiene todas las funciones encargadas de la configuración de cada uno de los sistemas que lo componen, el tiempo empleado para el despliegue es relativamente alto. Los procesos se ejecutan secuencialmente y en la misma terminal de comandos, por lo que el despliegue, especialmente durante el diseño y el período de pruebas del mismo, se hace excesivamente lento. Sin embargo, si la práctica se llevara a cabo en una plataforma como Amazon AWS o Google Cloud, y teniendo en cuenta que cada máquina virtual se podría ejecutar de manera independiente, la configuración de los elementos del escenario podría ser realizada simultáneamente de forma que se ahorrara un tiempo significativo en el despliegue.

Otro de los problemas detectados es la seguridad del escenario. Si bien es cierto que una correcta configuración del firewall garantiza protección frente a ciertos ataques, hoy en día son múltiples las posibles amenazas a las que se enfrentan aplicaciones como la desplegada. Una posible solución a esto sería la encriptación de la información mediante TLS, para proteger la información en tránsito entre máquinas virtuales. La generación de un par de claves (pública y privada) para encriptar la información, especialmente la más sensible, dotaría a la aplicación de una mayor seguridad frente a ataques como el de '*man in the middle*'.

Por último, habría que valorar la escalabilidad y flexibilidad de la aplicación. A pesar de que igual que se ha creado un cuarto servidor de aplicación podrían crearse más aún para satisfacer todo el tráfico necesario, el hecho de que las máquinas virtuales estén basadas en LXC hace que ofrezcan menores prestaciones en cuanto a memoria y rendimiento, pero sin embargo ante un número elevado de usuarios el escenario podría no responder de la manera esperada. En este sentido, para aumentar el número de servidores si es cierto que otras plataformas de contenedores ofrecen soluciones más inteligentes y flexibles, tanto para almacenar los contenedores como para administrar los entornos simulados.