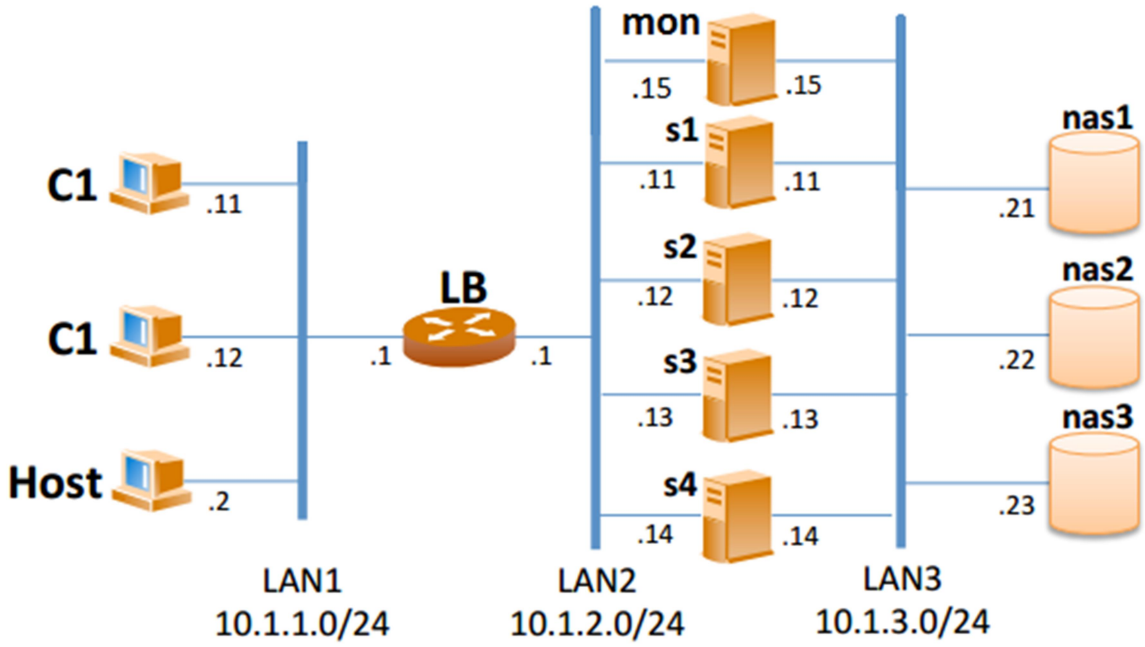
DISEÑO DE UN SISTEMA DE COMPARTICIÓN Y PRESENTACIÓN DE AUDIO ESCALABLE: CDPSfy

Diseño de la solución de alta disponibilidad: Arquitectura del sistema. Componentes y servicios que se ejecutan en cada máquina

La arquitectura del sistema es la que sigue:



-En el balanceador de tráfico LB se ejecutará el programa “Crossroads”, para la configuración del mismo, aplicando el algoritmo “Round Robin”.
-En el servidor web s1 se desplegará la aplicación CDPSfy
-En los servidores web s2, s3 y s4 se desplegará la aplicación “Tracks”
-En la MV “mon” se ejecutará Nagios para monitorizar el sistema
-En los servidores de disco nas1, nas2 y nas3 se montará un sistema de discos GlusterFS y se guardarán las canciones en el directorio /nas/canciones y las covers en el directorio /nas/caratulas.

La solución adoptada será escalable, pues tan solo haría falta añadir máquinas virtuales definiéndolas en el archivo de configuración xml, tanto si fueran para CDPSfy (como s1) como para Tracks (como s2, s3 o s4), y el tráfico se balancearía entre ellas gracias a LB. Así los nuevos clientes dispondrían de más recursos de computación. También se podrían añadir más discos al gluster para aumentar el almacenamiento.

Pasos seguidos en la consecución de la solución

Para poder diseñar un sistema con la arquitectura descrita, era necesario modificar la aplicación en nodejs-express "CDPSfy" que se nos proporcionaba para conseguir que ésta sirviera los ficheros de audio y los enviara a los discos del glúster de almacenamiento en vez de al directorio estático del servidor web. Para ello era necesario crear una nueva aplicación nodejs en los servidores de almacenamiento que recogiera las peticiones y las canciones que envía CDPSfy y las dirigiera a los discos de almacenamiento. Es la llamada "tracks", como también hemos descrito arriba.

Por tanto, nuestros primeros pasos fueron programar estas aplicaciones para que realizaran la labor deseada. Como era más rápido y se podía hacer inmediatamente sin instalar más que node, hicimos esta programación y pruebas en local. Simplemente arrancábamos la aplicación con "npm start" o "node index.js", cada una de ellas en diferente puerto (Con el comando export PORT=8181, por ejemplo) y modificando las direcciones de entrega de peticiones REST por 127.0.0.1:8080 (La dirección de loopback de las máquinas), para que fuera accesible desde el navegador web. También cambiamos la carpeta de los ficheros de audio por una local, en las direcciones de tracks. Básicamente desplegamos las dos aplicaciones en nuestro equipo.

- Para CDPSfy, tuvimos que modificar los métodos "create" para añadir la canción al sistema de archivos y no sólo sus metadatos; y "destroy" para eliminarla del sistema de archivos. Para ambos tuvimos que manejar los ficheros de audio del tipo multipart (pensado para dividir archivos grandes en varios bloques) proporcionados por el formulario de subida y tratados por el middleware "multer". Utilizamos para ello otro middleware llamado "needle" (https://github.com/tomas/needle), que maneja este tipo de archivos realizando peticiones REST en varios bloques. Dirigimos los posts de "create" a Tracks de esta manera, guardando la canción por una id y conservando su nombre original para mostrarlo al usuario; así como los delete en "destroy" para eliminarlas.

  Como mejoras adicionales, los ids de las canciones los creamos en orden y a continuación del último existente (Empezando en el número 5) y añadimos "(1)" tantas veces como sea necesario tras el nombre de la canción mostrada al usuario para que a éste no le aparezcan nombres de canciones repetidos en el interfaz. También programamos un script en JS que comprueba que el formato de canción subida se ajusta a los admitidos y que efectivamente se ha subido una canción (ya que la aplicación original daba una página de error si se daba a continuar sin subir ninguna canción).

  Una vez comprobado que esto funcionaba, como mejora también introducimos la posibilidad de subir covers (carátulas, imágenes) asociadas a cada canción, con su misma id y que se guardará en otra carpeta diferente. Para ello simplemente usamos

el mismo esquema que para las canciones, modificando también los scripts JS y las vistas ejs, teniendo en cuenta que en “destroy” no hace falta enviar dos peticiones, pues la id es la misma tanto para la canción como para la cover.

- Para Tracks, hemos creado desde cero una aplicación nodejs simple que sea capaz de recoger las peticiones REST que le llegan desde CDPSfy y almacenarlas en /nas, que es donde estará el gluster de archivos. Tratamos en diferentes funciones del controlador las peticiones de crear o proporcionar covers y canciones, pero no los delete por lo dicho arriba.

Una vez verificado que las aplicaciones funcionaban correctamente, hemos cambiado las rutas por las adecuadas para las MVs y hemos procedido a desplegar el escenario, arrancando las mismas y configurando GlusterFS de la manera indicada en la memoria (la cual comprobaremos mediante las capturas del siguiente apartado). Como este procedimiento es automatizable, hemos optado por realizar scripts como mejora que efectivamente automaticen completamente este despliegue. Primero arrancando el escenario con “start.sh” y luego realizando la configuración de cada MV con “setup.sh”. Con esto también tenemos configurado GlusterFS y Crossroads y traducidos los nombres de los dominios de la aplicación por las direcciones IP correspondientes. Además, hemos creado sendos repositorios en el servicio “Bitbucket” (parecido a Github, pero permite crear proyectos privados de forma gratuita) con CDPSfy y Tracks, que se instalan automáticamente en las MVs requeridas con estos scripts.

Hará falta crear una nueva MV para instalar Nagios y monitorizar el sistema. La hemos definido previamente en el xml de configuración de p7. Dentro de esta máquina, llamada “mon”, procederemos a instalar nagios de la forma indicada en la práctica 6 y automatizamos la monitorización dentro de ella con los scripts “mon.sh” y “nagios.sh”, que simplemente copian ficheros de configuración y reinician Nagios y Apache. La monitorización se termina de configurar con otro script llamado “monitorización.sh”.

Finalmente, sólo restó comprobar que la aplicación funcionaba según se requería.

Configuraciones realizadas

Las configuraciones están descritas en los scripts de configuración que hemos programado. Son los que siguen:

Arranque del escenario y envío de los scripts de configuración a cada máquina en start.sh:

```
#!/usr/bin/bash

bin/prepare-p7-vm

sleep 10

sudo vnx -f p7.xml -v -t

sleep 10

sudo cp scripts/s1.sh /var/lib/lxc/s1/rootfs/tmp
sudo cp scripts/s2.sh /var/lib/lxc/s2/rootfs/tmp
sudo cp scripts/s3.sh /var/lib/lxc/s3/rootfs/tmp
sudo cp scripts/s4.sh /var/lib/lxc/s4/rootfs/tmp
sudo cp scripts/lb.sh /var/lib/lxc/lb/rootfs/tmp
sudo cp scripts/nas1.sh /var/lib/lxc/nas1/rootfs/tmp
sudo cp scripts/nas2.sh /var/lib/lxc/nas2/rootfs/tmp
sudo cp scripts/nas3.sh /var/lib/lxc/nas3/rootfs/tmp
sudo cp scripts/mon.sh /var/lib/lxc/mon/rootfs/tmp
sudo cp scripts/nagios.sh /var/lib/lxc/mon/rootfs/tmp
sudo cp scripts/c1.sh /var/lib/lxc/c1/rootfs/tmp
sudo cp scripts/c2.sh /var/lib/lxc/c2/rootfs/tmp
```

Autenticación en cada máquina y ejecución de los scripts en setup.sh:

```
#!/usr/bin/bash

sudo apt-get install sshpass -y

sshpass -p 'xxxx' ssh -oStrictHostKeyChecking=no root@nas1 'sh /tmp/nas1.sh'
sshpass -p 'xxxx' ssh -oStrictHostKeyChecking=no root@mon 'sh /tmp/mon.sh' &
sshpass -p 'xxxx' ssh -oStrictHostKeyChecking=no root@s2 'sh /tmp/s2.sh' &
sshpass -p 'xxxx' ssh -oStrictHostKeyChecking=no root@s3 'sh /tmp/s3.sh' &
sshpass -p 'xxxx' ssh -oStrictHostKeyChecking=no root@s4 'sh /tmp/s4.sh' &
```

```
sshpass -p 'xxxx' ssh -oStrictHostKeyChecking=no root@s1 'sh /tmp/s1.sh' &
sshpass -p 'xxxx' ssh -oStrictHostKeyChecking=no root@lb 'sh /tmp/lb.sh' &
sshpass -p 'xxxx' ssh -oStrictHostKeyChecking=no root@c1 'sh /tmp/c1.sh'
sshpass -p 'xxxx' ssh -oStrictHostKeyChecking=no root@c2 'sh /tmp/c2.sh'
```

Configuración del balanceador de tráfico LB con Crossroads:

```
#!/usr/bin/bash

sudo apt-get update -y > /dev/null

xr -dr --verbose --server tcp:0:80 --backend 10.1.2.11:80 --web-interface 0:8001 &

xr -dr --verbose --server tcp:0:8080 --backend 10.1.2.12:8080 --backend 10.1.2.13:8080 --
backend 10.1.2.14:8080 --web-interface 0:8002
```

Configuración del Gluster con GlusterFS en nas1:

```
#!/usr/bin/bash

gluster peer probe 10.1.3.22
sleep 2
gluster peer probe 10.1.3.23
sleep 5
gluster peer status
gluster volume create nas replica 3 10.1.3.21:/nas 10.1.3.22:/nas 10.1.3.23:/nas force
sleep 5
gluster volume start nas
sleep 5
gluster volume info

mkdir /nas/canciones
mkdir /nas/caratulas
```

Comprobación del glúster en nas2 y nas3:

```
#!/usr/bin/bash
```

```
gluster peer status
gluster volume info
```

Configuración del archivo hosts en los clientes c1 y c2 para la traducción de los dominios:

```
#!/usr/bin/bash

sudo apt-get update -y > /dev/null
sudo echo "10.1.1.1 cdpsfy.es" >> /etc/hosts
sudo echo "10.1.1.1 tracks.cdpsfy.es" >> /etc/hosts
```

Despliegue de CDPSfy en el servidor s1:

```
#!/usr/bin/bash

sudo apt-get update -y > /dev/null
sudo apt-get install nodejs -y
sudo echo "10.1.2.1 tracks.cdpsfy.es" >> /etc/hosts
export PORT=80
git clone https://sgrcdps:cdps2016@bitbucket.org/sgrcdps/cdpsfy.git
cd cdpsfy
npm install
npm start
```

Despliegue de Tracks en el servidor de discos s2:

```
#!/usr/bin/bash

sudo apt-get update -y > /dev/null
sudo apt-get install nodejs -y
mkdir /mnt/nas
mount -t glusterfs 10.1.3.21:/nas /mnt/nas
sleep 5
mkdir /mnt/nas/canciones
mkdir /mnt/nas/caratulas
git clone https://sgrcdps:cdps2016@bitbucket.org/david_santosg/tracks.git
cd tracks
npm install
```

```
node index.js
```

Y en los otros dos servidores de disco, s3 y s4, sin necesidad de crear las carpetas:

```
#!/usr/bin/bash

sudo apt-get update -y > /dev/null
sudo apt-get install nodejs -y
mkdir /mnt/nas
mount -t glusterfs 10.1.3.22:/nas /mnt/nas
git clone https://sgrcdps:cdps2016@bitbucket.org/david_santosg/tracks.git
cd tracks
npm install
node index.js
```

Configuración de Nagios según lo descrito en el apartado anterior. Mon.sh:

```
#!/usr/bin/bash

sudo apt-get update -y > /dev/null
```

Configuración de Nagios según lo descrito en el apartado anterior. Nagios.sh:

```
#!/usr/bin/bash

sudo service nagios3 restart
sudo service apache2 restart
```

Configuración de Nagios según lo descrito en el apartado anterior. Monitorización.sh:

```
#!/usr/bin/bash

sudo cp nagiosconfig/*.cfg /var/lib/lxc/mon/rootfs/etc/nagios3/conf.d/

sshpass -p 'xxxx' ssh -oStrictHostKeyChecking=no root@mon 'sh /tmp/nagios.sh'
```

Despliegue en OpenStack y AWS

Para replicar el despliegue de este escenario en AWS u OpenStack, habría que crear una correspondencia entre las máquinas virtuales que hospedan las aplicaciones y servicios, y las unidades de cómputo, almacenamiento y red disponibles en dichas plataformas.

De esta forma, el balanceador y los servidores web pueden desplegarse mediante instancias que ejecutan las aplicaciones usadas a lo largo de la práctica, el almacenamiento puede contratarse según las prestaciones necesarias (Glacier), y las redes pueden configurarse como VPNs entre las instancias, añadiendo IPs elásticas para las instancias que hospedan los servicios visibles desde el exterior.

Aplicaciones CDPSfy, Tracks y scripts de automatización

Se aportan en el fichero zip que se adjunta a la entrega de esta memoria. Las mejoras realizadas respecto a la práctica base están descritas a lo largo de esta memoria.