Centro de Datos y de Provisión de Servicios

Práctica final 2: Despliegue de una aplicación escalable

Quiz 2019

17 Enero 2019

Pablo Caraballo Llorente
Mario Penavades Suárez

Tabla de contenidos

Introducción

Este proyecto integra los conocimientos adquiridos a lo largo de la asignatura relativos a virtualización, almacenaje de datos replicado y demás competencias necesarias en un centro de datos.

El sistema básico cuenta con un firewall, un balanceador tráfico, 3 servidores web, 3 sistemas de almacenamiento NAS sincronizados y una base de datos SQL.

El funcionamiento de los elementos es el siguiente:

- El firewall, solo permite el acceso mediante ping y al puerto 80 de TCP de la dirección del balanceador de tráfico. El resto del tráfico se deniega.
- El balanceador de tráfico se encarga de balancear entre todos los servidores utilizando round-robin.
- El cluster de almacenamiento, guarda las imágenes de los quizzes. Y agrupa a los 3 sistemas de almacenamiento NAS.
- La información manejada por el quiz se almacena en un servidor de bases de datos utilizando MariaDB.
- Los servidores contienen la aplicación Quiz que la hacen accesible a través de Internet. Hay 3 servidores cuya carga se verá administrada por el balanceador previo.

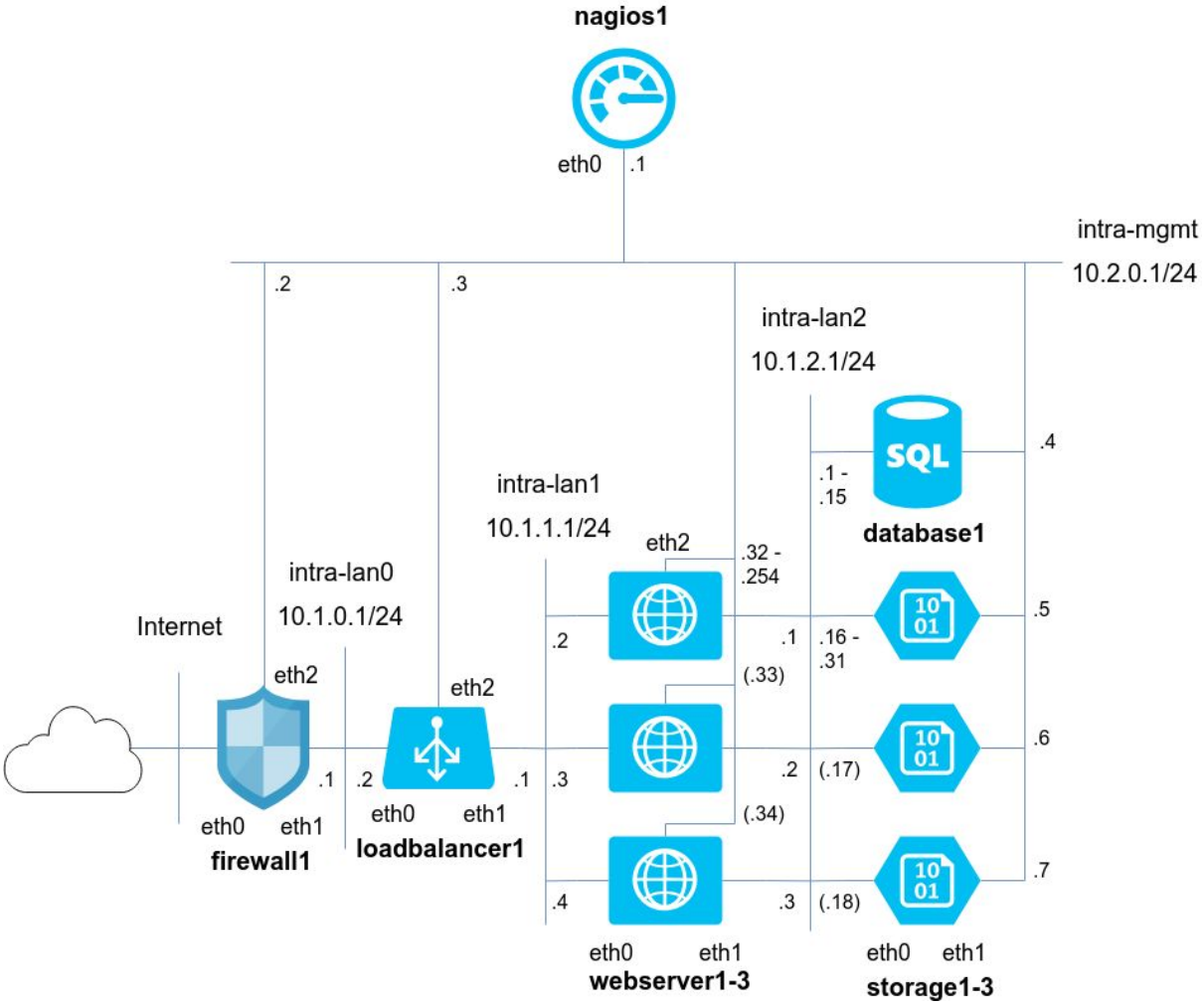
Para la realización de esta práctica hemos utilizado Linux Containers mediante LXD.

A continuación se muestra un diagrama del escenario de red resultante en caso de ejecutar cualquiera de todas las mejoras posibles, o sólo el escenario básico (omitiéndose los componentes correspondientes en cada caso). Como se observa existen 4 redes principales:

- intra-lan0: es la primera red interna tras pasar el firewall. Sólo contiene al cortafuegos y al balanceador de carga.
- intra-lan1: será la red que contiene la segunda interfaz del balanceador de carga y los servidores a los que se distribuyen las peticiones.
- intra-lan2: será la última red de aplicación, que habilita la conexión entre el sistema de ficheros replicado con el servicio quiz y la conexión de la base de datos con los mismos. Como se observa, se asignan rangos de direcciones suficientes a cada tipo de servidor con el fin de que en futuros incrementos de réplicas no haya solapamientos.
- intra-mgmt: esta será la red de mantenimiento en caso de habilitar la monitorización por nagios. Para esta red se usa un “*DHCP*” que asigna direcciones estáticas a los hosts según se van conectando a la red. La única condición es que el servidor con nagios tenga la primera dirección.

Finalmente cabe destacar que en ninguna red de las anteriormente mencionadas contiene un gateway o router. Esto es porque las redes al ser pequeñas se manejan end to end y porque en ningún caso va a existir conexiones entre subredes distintas, lo que hace que no se necesite un elemento enrutador.

Por otra parte, al estar cada red contenida en un bridge distinto hace que un atacante que se haga con el control de uno de los servidores no pueda saltar a otra subred con sólo cambiar su configuración de red interna.



Uso

El mecanismo se puede analizar como una estructura de 3 componentes:

lxc-setup.sh

Es el fichero que se encarga de ejecutar el escenario para esta práctica. Necesita ser ejecutado como root ya que utiliza comandos red que lo requieren.

Para poder ejecutarlo habrá que ejecutar sobre un terminal, **dentro de `bin/`** el comando:

```
sudo lxc-setup.sh
```

Nota: Se asume que la versión instalada de Python es 2.7 y que tanto lxc como lxd están instalados en el sistema host.

En primer lugar, este fichero hará un control sobre las dependencias mínimas necesarias para poder ejecutar la práctica, así como de los paquetes python necesarios:

- `net-tools`: para poder usar ifconfig
- `bridge-utils`: para poder crear bridges virtuales que separen las redes de los contenedores.
- Paquete `pybrctl`: para manejo de `brctl` por python.
- Paquete `yaml`: para poder interpretar YAML cómodamente en python.

Después recorrerá la lista de scripts que configuren las plantillas seleccionadas mediante `lxc-setup.py`.

lxc-setup.py

Es el fichero python que contiene el script para configurar cada máquina. Es llamado por el fichero anterior `lxc-setup.sh` (o desde consola) tantas veces como máquinas de categoría distinta hay, es decir: firewall, balanceador de tráfico, webserver, sistema de almacenamiento nas y sistema de base de datos. Se encargará de instalar y configurar todos los parámetros indicados en el fichero `config-file.json` de cada clase, de modo similar a como se haría con Docker/ Docker-Compose.

Para poder ejecutarlo habrá que ejecutar sobre un terminal el comando:

```
python lxc-setup.py [config-file.json]
```

[name]-cfg.json (config-file)

Es el fichero que describe el contenedor que se desea crear. Tiene una estructura similar al Dockerfile junto a Docker-Compose. En él se definen los siguientes campos:

- image: String, se trata de la imagen lxc que va a usar, del modo `[<remote>]:<image>`.
- name: String, será el nombre que se le de al contenedor.
- dependencies: [String], aquellos programas que necesitemos presentes en el contenedor.
- interfaces: [{name: String, network: String, address: String, gateway: String}], configuración personalizada sobre las interfaces de red. Esta configuración se hace de manera interna con `ifupdown` y externa con `brctl`:
    1. name será el nombre de la interfaz (eth0, eno1...)
    2. network indica a qué red queremos conectarla (intra-lan0, intra-mgmt...)
    3. address será la dirección IP que queremos en el contenedor, tipo CIDR (AA.AA.AA.AA/MM)
    4. gateway será el gateway de la red, en caso de haberlo.
- forwarding: Boolean, si queremos que el contenedor reenvíe paquetes IP. Esto provoca un log de tipo WARNING recordando que dicho contenedor tiene el forward activado.
- run: [String], al igual que en Docker son comandos que se ejecutan antes de finalizar la instalación **en cada contenedor del config-file.**
- cmd: [String], son comandos que se ejecutan *on-boot* del contenedor. `lxc-setup.py` se encarga de generar un fichero en shell con la colección de comandos y de importarlo en `/etc/init.d/` para ello. **Se hace para todos los contenedores del config-file.**
- env: {String: String}, variables de entorno que se agregan al `.bashrc` en orden de que siempre estén presentes. **Se agregan a todos los contenedores del config-file.**
- scripts: [String], se trata de scripts shell literales que se ejecutarán tras la instalación del `config-file`. Esto nace de la necesidad de particularizar ciertos comandos **a algún componente de la réplica en vez de a todos**.
- replication: Int, número de réplicas. **Tanto si es 0, como 1, como si no existe, se genera un solo contenedor cuyo nombre final será `[name]1`.**

Puntos débiles del sistema

Los puntos débiles del sistema son, entre otros, los siguientes:

- Solo hay una máquina donde está alojado el sistema de base de datos, por lo que en caso de fallo en esta máquina, el servicio quedaría totalmente inutilizado. Es por ello que hemos configurado una segunda máquina que replica el sistema de base de datos de la primera.

- Igualmente que el anterior, solo hay un servidor donde está alojado el balanceador de carga y el firewall, por lo que en caso de fallo, el servicio quedaría totalmente inutilizado. Podría configurarse un balanceador de tráfico más al igual que un firewall más.

- Respecto a la mejora de monitorización, lo ideal sería una implementación como la vista en clase: primero hacer una etapa previa de monitorización de cada grupo de servidores que sirven la misma función y después una monitorización de los monitores de la primera fase.

- Para aumentar más la seguridad, sería dividir el servicio en zonas. Una zona desmilitarizada para los servidores donde se aloja la aplicación y una zona totalmente protegida para la base de datos y los sistemas de almacenamiento. A esto se le podría agregar un servicio de VPN con autenticación por clave RSA, de tal modo que el acceso al servidor de monitoreo no sería público y se podrían realizar tareas de mantenimiento desde fuera de la red. Además como la autenticación es por par de claves y no por contraseña alfanumérica la seguridad del proceso se garantiza.

- A lo largo de toda la práctica no se comenta en ningún lugar la importancia de hacer procesos automatizados en el arranque del sistema. Cabe recordar la importancia de esto ya que en caso de tener más sistemas, si uno cae y lo levantamos o si necesitamos reiniciar cualquier servidor, con procesos que se inicien en arranque no tenemos que acordarnos cual es la función que sirve cada uno. Con nuestro mecanismo esto se solventa, en parte, añadiendo los comandos necesarios en el atributo `cmd` de la plantilla de configuración.

- El sistema de bridges por el que se levanta el escenario es tal que si se reinicia el servidor host en el que se ha levantado, se borran los bridges virtuales creados.

- El balanceador de carga se habilita para que distribuya el tráfico entre unos sistemas, independientemente de la aplicación que estos sirvan. Esto hace que si un atacante consigue acceder a un webserver, pueda cambiar el servicio que éste presta sin que nos demos cuenta.

Mejoras realizadas

Las mejoras funcionales realizadas, están incluidas en el sistema final, por lo que cuando se ejecute el sistema, se hará con todas las mejoras funcionales incluidas.

Contenedores

Se ha realizado la implementación de todo el escenario mediante Linux Containers puesto que dan mayor rapidez para su creación, y la posibilidad de destruirlos de forma casi inmediata.

En la entrega se proponen dos escenarios:

- `bin/lxc-setup.sh - Escenario base sin servidor web adicional.`
- `templates/webapp/webserver-add-cfg.json - config-file que agrega el cuarto servidor web.`

En ambos casos el script funciona del mismo modo: primero hace el control de dependencias y luego llama sucesivamente al programa en python configurando las plantillas correspondientes.

Para la realización de la prueba oral el procedimiento consiste en ejecutar la siguiente secuencia:

```
sudo ./lxc-setup.sh
sudo python lxc-setup.py ../templates/webapp/webserver-add-cfg.json
```

El primero es el que genera todo el escenario con las mejoras implementadas, exceptuando el cuarto servidor web. El segundo se ejecutará con el profesor para mostrar el funcionamiento del script, completando todas las mejoras presentadas.

AWS

Para la implementación de este servicio en la nube, como puede ser Amazon Web Services habrá que realizar las siguientes configuraciones:

- **Firewall**: Se deberá crear una instancia para el Firewall. Amazon Web Application Firewall [1] permite controlar el tráfico que se desea habilitar o bloquear en las aplicaciones web mediante la definición de reglas de seguridad web personalizables. Que serán las proporcionadas en el enunciado.

- **Balanceador**: Amazon proporciona sus propios Balanceadores de tráfico, Amazon Elastic Load Balancing [2]. Este Balanceador distribuye automáticamente el tráfico de aplicaciones entrantes a través de varios destinos, tales como instancias de Amazon EC2, que serán los servidores necesarios en nuestro escenario como explicamos a continuación. Al crear la instancia en Amazon Elastic Load Balancing, se proporcionarán los destinos.

- **Servidores**: Amazon proporciona un servicio web que facilita capacidad informática en la nube mediante Amazon EC2 [3]. Se deberá crear una instancia EC2 por cada servidor, es decir, 3 instancias. En caso de que se realice el supuesto caso opcional se

deberán crear 4 instancias y se deberán configurar correctamente. También existe otra opción que se verá más adelante.

- **Base de datos**: Amazon proporciona también sistemas de bases de datos como es Amazon Relational Database Service [4]. Se deberá crear una instancia y realizar la configuración necesaria, para poder acceder desde los servidores. También se permite el uso de MariaDB. El servicio suministra capacidad rentable y escalable al mismo tiempo que automatiza las arduas tareas administrativas, como el aprovisionamiento de hardware, la configuración de bases de datos, la implementación de parches y la creación de copias de seguridad. Lo que se adapta a la perfección a nuestro escenario.

- **Gluster/NAS**: Amazon proporciona, entre otros muchos sistemas, instancias para almacenamiento. Amazon Elastic File System [5] suministra un almacenamiento de archivos simple, escalable y elástico para utilizar con los servicios en la nube de AWS y los recursos locales. Se montan en instancias Amazon EC2, que serán los NAS necesarios.

Para implementar las mejoras, deberemos añadir los siguientes elementos:

- **Nagios**: Amazon nos facilita la monitorización gracias a la posible incorporación de Nagios en sus instancias. Por lo que para la monitorización de nuestro escenario también se podrá aplicar AWS.

- **Servidor Extra**: Amazon ofrece la posibilidad de crear Auto Scaling Group [6], grupos en los que se especifica un número mínimo y máximo de instancias por lo que, si el sistema cree conveniente arrancar una nueva instancia, se hará sin problemas. Tal y como deja crear, también proporciona la eliminación, por lo que en momentos de mucha carga se podría crear un nuevo servidor y tras ello, destruirlo.

Todos estos elementos se deberán configurar de la misma manera que están configurados mediante los scripts correspondientes a cada elemento.

Balanceador de tráfico

Se ha realizado la configuración del algoritmo Round-robin con pesos [7]. Se ha decidido aplicar a los dos primeros servidores unos pesos del 35% para que sean estos, los que más tráfico absorban y para los dos últimos servidores, en caso de que estén disponibles los dos, unos pesos del 15% para que absorban menos tráfico. En todos los casos se habilita el modo check, así, si cae algún servidor se omite en la lista de posibles destinos de peticiones. Con esto conseguimos también que un mismo script sea válido para distintas configuraciones de servidores (unos levantados y otros no).

Consolidación de los logs

Se ha realizado la consolidación de los logs de todos los servidores mediante el volcado de todos ellos en un fichero en la máquina host. En este fichero queda grabado todo lo que se realiza en el sistema con la fecha, la hora y el sistema desde el que se realiza cualquier

acción. Se ha implementado gracias a la librería logging de Python. El fichero donde se guardan los logs del proceso de instalación es:

`logs/registro.log`

Replicación de bases de datos

Se ha realizado la replicación del sistema de base de datos en una segunda máquina para aumentar la fiabilidad del sistema en caso de fallo de la única máquina que había disponible. Se ha replicado en otra máquina con MariaDB, utilizando el sistema Maestro-Esclavo [8], donde el esclavo sólo se permite leer, puesto que es un sistema de replicación temporal. El fichero de configuración donde se muestra dicha mejora es:

`templates/database/database-upgrade-cfg.json`

Nagios para monitorización

Se establece una opción en el fichero de configuración llamada nagios . Esta opción, de tipo booleana, puesta a true hace que durante el proceso de creación del contenedor, se registren sus datos como host dentro del servidor que está corriendo nagios core. Por otra parte hace que en dicho contenedor a ser administrado se instalen los plugins y dependencias necesarias para ser monitorizado por Nagios [9] [10]. Para que esta funcionalidad sea efectiva, habrá que haber creado previamente un contenedor nagios. Por eso ya se ha creado una plantilla que lo levanta:

`templates/management/nagios-cfg.json`

Para esta mejora se establece una nueva regla en el firewall de tal manera que permita el acceso al servidor de mantenimiento (`nagios1`) mediante el puerto 3030 del mismo. Por lo que podemos acceder al monitoreo de los sistemas mediante la url:

`http://[IP_EXTERNA_FIREWALL]:3030/nagios`

Los credenciales de acceso serán: `nagiosadmin/admin.`

A continuación se muestra la configuración resultante del firewall. Destacamos la NAT que se hace para los servicios del Quiz y del Nagios de modo que un cliente ajeno a la organización, que sólo conozca la dirección pública, pueda acceder a los dos servicios. Mientras que al servicio Quiz se puede acceder sin problema, para el nagios se requiere identificación, con lo que se controla que dicho cliente no pueda ver el estado interno de los sistemas, pero sí un trabajador que esté fuera de la oficina.

0	nagios	Any	Any	mgmt	Both	Accept	Any	log	
1	Any	nagios	nagios http https	outside	Both	Accept	Any	log	
2	Any	Any	any ICMP	outside inside	Both	Accept	Any	log	
3	Any	loadbalancer	http https	outside	Both	Accept	Any	log	
4	Any	Any	Any	Any	Both	Deny	Any	log	

0	Any	Any	nagios	self_mgmt	nagios	http	Auto	Auto	Translate
1	Any	Any	http https	self	loadbalancer	Original	Auto	Auto	Translate

Añadir y configurar nuevo servidor dinámicamente

Se ha realizado la implementación del nuevo servidor como una extensión y de la misma manera a como se realiza con los otros 3 servidores de la parte obligatoria. Para eso sólo hay que incrementar el valor de replicación en la plantilla de los webservers (`templates/webapp/webserver-upgrade-cfg.json`) y añadir una línea en el atributo `scripts` de la plantilla, que monte un NAS en la carpeta de fotos.

En caso de no haber este cuarto servidor, el balanceador de tráfico balanceará entre los 3 servidores obligatorios y cuando este 4º entre en acción, balanceará entre los 4 servidores disponibles.

En caso de que se quiera añadir un N-Servidor web a la estructura (con N < 15 a fin de evitar solapamientos de direcciones red) sólo hay que duplicar la plantilla del webserver, restringiendo la replicación a 0, 1 u omitiéndola y finalmente teniendo en cuenta el montaje de los servidores NAS en el apartado `scripts`, de modo que se ajuste a los servidores de almacenamiento disponibles. Se adjunta una plantilla que agregaría el cuarto servidor en caliente:

`templates/webapp/webserver-add-cfg.json`

Por lo tanto se proponen dos métodos al usuario

1. Generar el escenario desde un inicio con el cuarto servidor web mediante el uso único de la plantilla `webserver-upgrade-cfg.json`.
2. Generar el escenario inicial con 3 servidores web con la plantilla `webserver-cfg.json` y posteriormente extender la replicación del servicio quiz con un cuarto servidor usando la plantilla `webserver-add-cfg.json`

Referencias

[1] Amazon Web Application Firewall - https://aws.amazon.com/es/waf/

[2] Amazon Elastic Load Balancing - https://aws.amazon.com/es/elasticloadbalancing/

[3] Amazon Elastic Compute Cloud - https://aws.amazon.com/es/ec2/

[4] Amazon Relation Database System - https://aws.amazon.com/es/rds/

[5] Amazon Elastic File System - https://aws.amazon.com/es/efs/

[6] Amazon Auto Scaling Group - https://aws.amazon.com/es/autoscaling/

[7] Load Balancing with HAProxy - https://serversforhackers.com/c/load-balancing-with-haproxy

[8] Chapter 17 Replication MariaDB - https://dev.mysql.com/doc/refman/8.0/en/replication.html

[9] Nagios Core - Installing Nagios Core From Source - https://support.nagios.com/kb/article/nagios-core-installing-nagios-core-from-source-96.html#Ubuntu

[10] Nagios Client or NRPE - Installation and Configuration on Centos 7 - https://www.youtube.com/watch?v=wnJ2KL93jwQ&t=346s