Práctica 6 – Servicio balanceado con redundancia

1. Introducción y objetivos

El objetivo de esta práctica es construir un escenario donde se ofrezca un servicio web balanceado repartido entre múltiples hosts virtuales con protección en varios niveles.

Esta práctica recoge conceptos vistos en actividades anteriores sobre máquinas virtuales, contenedores e interconexión de redes de contenedores en diferentes VMs empleando VXLAN. Se añade a todo lo anterior un ejemplo de balanceador en capa 7, redundancia de primer salto mediante VRRP y protección en capa 2 mediante STP (estos dos últimos aspectos están vistos en asignaturas previas).

Se recomienda que guarde una copia en un fichero de texto de los comandos que va ejecutando en cada VM por si tuviera que rehacerlos y para que posteriormente pueda revisar todo lo que ha tenido que configurar.

2. Topología capa 3 y aplicación

La Figura 1 muestra el escenario objetivo visto en el nivel IP. Existen dos subredes IP:

- Subred interna: 10.0.0.0/24. En ella se encuentran los servidores web (serv1, serv2 y serv3). Todos ofrecen el mismo contenido. En ella se encuentran también los interfaces internos de los dos balanceadores (bal1 y bal2).
- Subred externa: 172.16.1.0/24. En ella están los usuarios, así como los interfaces externos de los balanceadores.

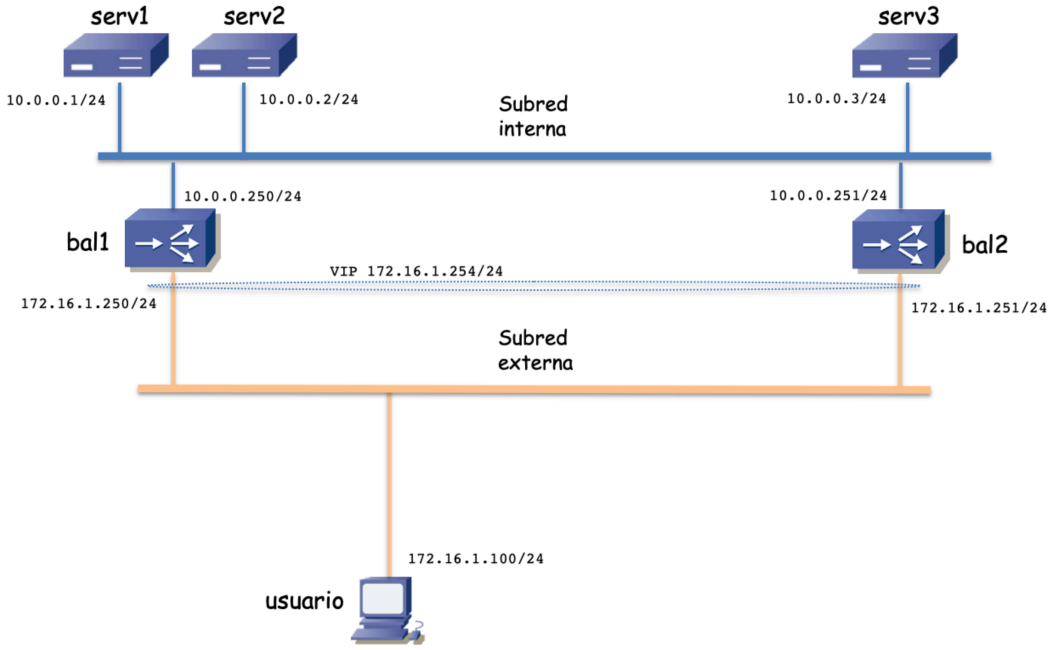

Figura 1 - Topología capa 3

Los dos balanceadores participan en un grupo VRRP protegiendo la dirección IP virtual 172.16.1.254 mediante la cual se ofrece el servicio web. Los balanceadores no enrutan ni natean el tráfico, así que los usuarios no pueden acceder directamente a los servidores. Estos balanceadores actuarán como proxy haciendo además de balanceador en capa 7. Es decir, dado el esquema de la Figura 1 los usuarios pueden acceder con su navegador web a http://172.16.1.254:8000; la conexión TCP se hará con el balanceador que actúe como maestro del grupo VRRP, el cual atenderá a la petición HTTP seleccionando un servidor

interno al que pedirle ese recurso por otra conexión TCP, en esta ocasión por el interfaz interno.

El balanceador bal1 actuará como el maestro del grupo VRRP y bal2 como backup, así que las peticiones las cursa bal1 salvo que éste falle. En caso de que bal1 pierda conectividad con la subred externa sería el protocolo VRRP quien se encargara de que bal2 pasara a atender a la dirección IP virtual (bal2 dejaría de recibir los mensajes periódicos de VRRP de bal1 y con eso decidiría declararse como maestro y hacer ARP gratuito de la IP virtual para avisar a los usuarios y los switches de la LAN externa de que su interfaz es quien atiende ahora a la dirección IP virtual).

Los balanceadores conocen la existencia de los 3 servidores, hacen comprobaciones periódicas de que pueden llegar a ellos y de que siguen funcionando correctamente (*health checks*) y reparten las peticiones entre ellos mediante un algoritmo simple de *round robin*.

En los servidores emplearemos Apache como servidor web, el cual entregará un index.php ligeramente diferente desde cada servidor para que podamos distinguir en el usuario qué servidor ha atendido a la petición y desde qué balanceador le llegó.

3. Topología capa 2

En la Figura 2 se han añadido los equipos capa 2 y los enlaces entre ellos. Se muestra por un lado la subred interna, que está formada por dos conmutadores, entre los cuales se encuentran repartidos tanto los servidores como los interfaces internos de los balanceadores. Por otro lado, la subred externa contiene cuatro conmutadores con redundancia de caminos entre ellos. La redundancia no es completa para simplificar el ejercicio pero al menos se podrá ver lo que sucede cuando fallan algunos enlaces o algunos equipos capa 2 de la subred externa.

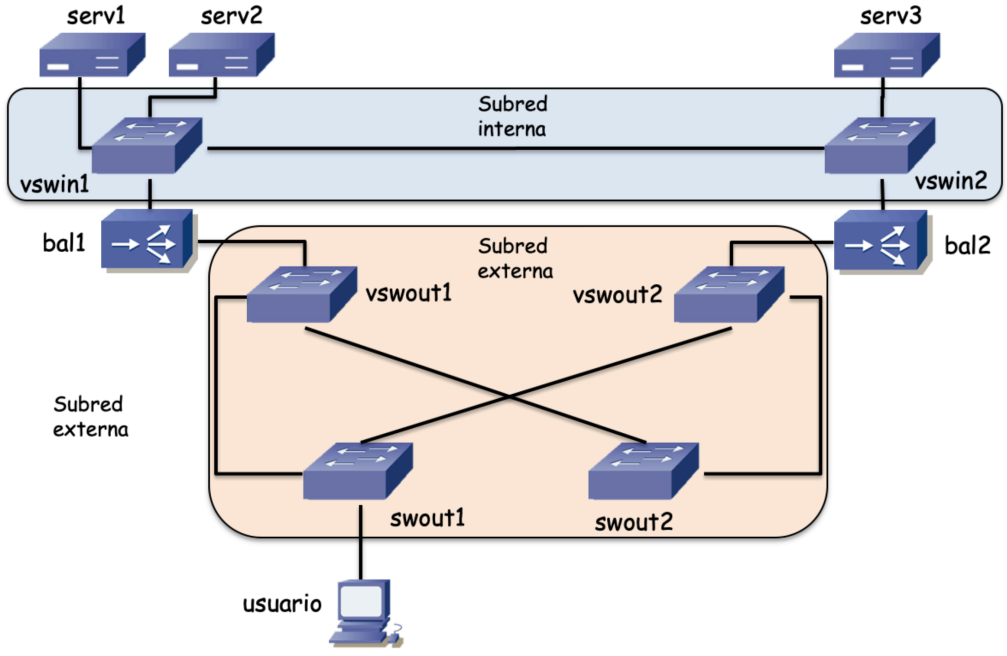

Figura 2 - Topología Ethernet

En la subred externa se empleará STP para eliminar los bucles. Tal y como es la topología podría fallar cualquier enlace entre los conmutadores de la subred externa y debería seguir funcionando la comunicación entre el usuario y el servicio pues existe un camino alternativo. Por ejemplo, en caso de fallo del conmutador vswout1 que da servicio a bal1 debería actuar VRRP pasando bal2 a ser el maestro, dado que dejaría de ver los mensajes VRRP de bal1.

Evidentemente si falla swout1 el usuario queda desconectado de la red. Si quiere puede crear otro usuario conectado a swout2 para comprobar que desde él sigue viéndose el servicio en funcionamiento.

4. Implementación real en esta actividad práctica

La Figura 3 muestra los equipos físicos que tendríamos realmente. Los conmutadores son conmutadores físicos y los dos hosts van a hospedar a los servidores y balanceadores empleando virtualización.

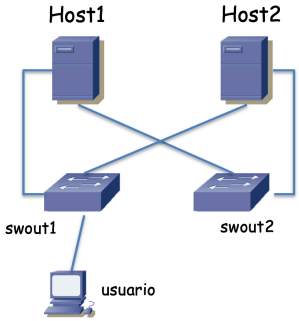

Figura 3 - Elementos físicos en un caso real

La Figura 4 representa los elementos en cada Host y cómo se va a llevar a cabo en prácticas la configuración. En el laboratorio podríamos emplear dos PCs para actuar como los dos hosts pero las máquinas de las que disponemos no están repartidas en diversos conmutadores ni tienen 2 NICs cada una; además sería un problema que probaran a estar desconectándolas. Se podría llevar a cabo la actividad empleando los armarios de prácticas, donde sí hay varios PCs y conmutadores, pero de momento no están preparados (por cuestiones de software) para ello.

Se ha decidido emplear máquinas virtuales para representar todos los elementos de la Figura 3 que serían equipos físicos reales. Tanto Host1 como Host2 serán VMs en VirtualBox. Se sugiere emplear para ellas una Ubuntu Server que al no tener interfaz gráfico requeriría menos de 1GB de RAM para cada una (en las pruebas se ha llevado a cabo esta práctica consumiendo cada una de estas dos VMs menos de 300MB de RAM).

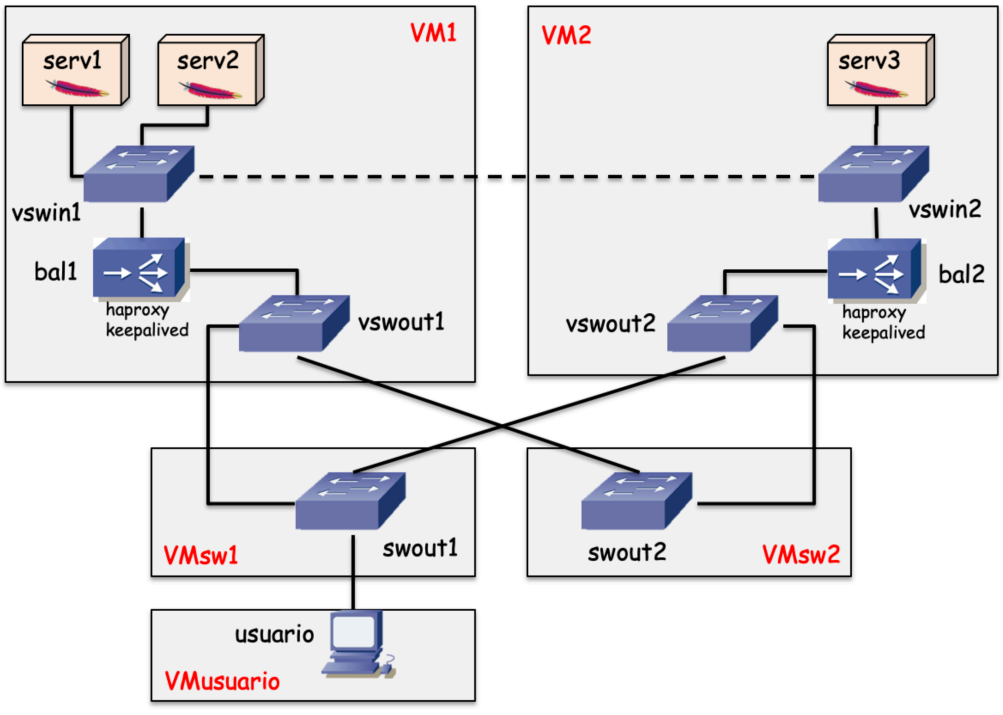

Figura 4 - Elementos implementados en VMs

Los conmutadores swout1 y swout2 que serían equipos físicos los implementaremos con dos VMs y empleando bridging en el kernel de Linux (para esto vale con una VM muy pequeña como se han empleado en prácticas anteriores). Esto nos permitirá por ejemplo emplear STP en estos puentes o ver el contenido de sus tablas de direcciones MAC, a diferencia de si empleáramos vSwitches de VirtualBox (*Internal Networks*) para implementar esos dos conmutadores. Los conmutadores vswout1, vswout2, vswin1 y vswin2 serán puentes creados en los kernel Linux de VM1 y VM2.

Finalmente, el usuario será una quinta VM. Aquí puede ser útil (aunque no imprescindible) una instalación con interfaz gráfico para poder ejecutar un navegador web y acceder al servicio. En prácticas anteriores se han empleado distribuciones pequeñas de Linux con interfaz gráfico que pueden servir para esta tarea (por ejemplo una Slitaz).

La Figura 5 resume cómo sería la interconexión entre las 5 VMs. Se emplearían *Internal Networks* de VirtualBox para cada uno de los enlaces entre VMs. Se han añadido interfaces NAT a VM1 y VM2 para permitir que estas VMs accedan a Internet y poder descargar software en ellas (por ejemplo para descargar imágenes de contenedores que necesitaremos o instalar software en ellos). Se han representado también interfaces al host (*vboxnet*) en las cuatro VMs que no tienen interfaz gráfico. Estos últimos interfaces son opcionales y nos permitirán hacer ssh desde el host que corre VirtualBox a cada una de esas cuatro VMs para configurarlas con mayor comodidad (aunque si se desea se puede trabajar directamente sobre ellas).

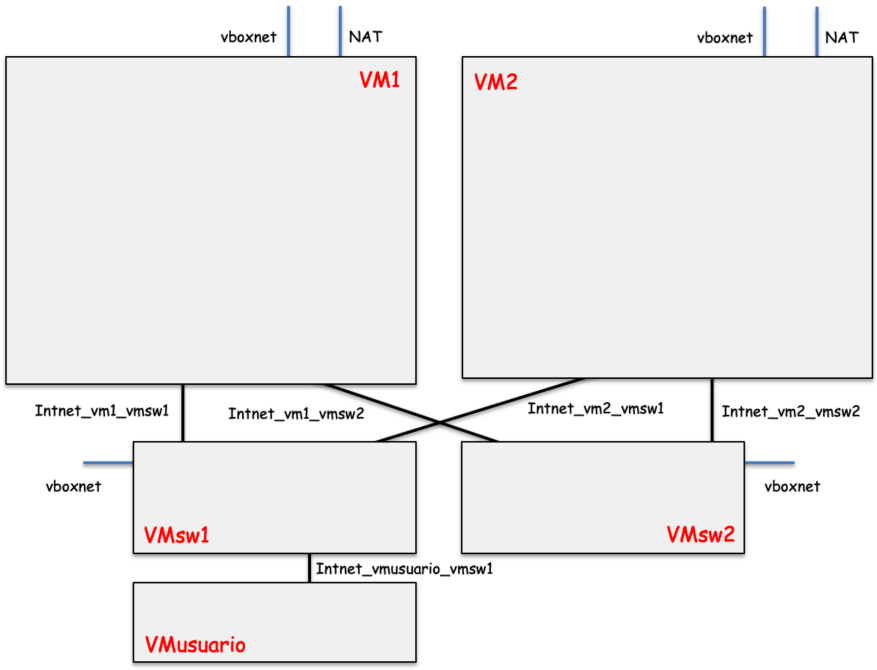

Figura 5 - Interconexión de VMs en VirtualBox

Una vez presentada una visión global de la actividad iremos construyendo y probando el funcionamiento de las diferentes partes del servicio hasta alcanzar su configuración completa.

5. Creación de la subred externa

Comenzaremos por construir la subred externa como se muestra en la Figura 6. Para ello cree las 5 VMs con todos sus interfaces tal y como se han descrito anteriormente. Permita el modo promiscuo en todos los interfaces de VMs en las cuales se vaya a configurar un bridge conectado a ese interfaz. Esta opción se encuentra entre las opciones avanzadas de la configuración de cada interfaz en VirtualBox y debe activarla antes de arrancar la VM.

Cree los 4 puentes (mediante *brctl*) y añada a ellos los interfaces correspondientes. Active STP (una opción de *brctl*) en los 4 puentes antes de activar (*ifconfig up*) los interfaces pues está creando bucles en capa 2.

Configure a la VM de usuario la dirección IP 172.16.1.100/24. Fíjese que ahora mismo no hay más interfaces IP en esa subred. Le daremos dirección IP en esta subred a VM1 y VM2 asignándola a los interfaces de sus puentes. Asigne 172.16.1.250 a vswout1 y 172.16.1.251 a vswout2. Ahora debería poder probar la comunicación IP entre esas 3 direcciones. De esta forma VM1 y VM2 tienen un interfaz IP en la subred externa. Los interfaces IP de VM1 y VM2 no están representados en la Figura 1; añádalos al dibujo si lo desea.

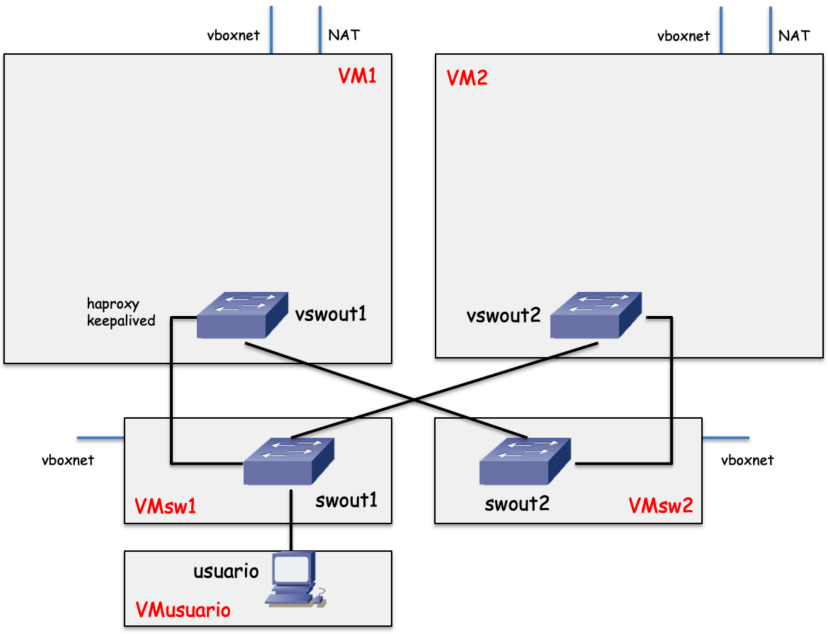

Figura 6 - Equipos en la subred externa

En los 4 puentes puede ver el estado y rol STP de los puertos (con opciones de *brctl*). Configure el puente swout1 como puente raíz asignándole un valor de prioridad menor que la que tienen el resto de puentes y observe si falla la comunicación mientras se produce el recálculo del árbol por este cambio, así como vigile los cambios de estado de los puertos de los puentes en ese proceso de transición.

Averigüe qué puerto ha quedado bloqueado por STP y verifique que coincide con lo que teóricamente debería suceder.

Pruebe a desactivar (*ifconfig down*) alguno de los interfaces en uso y que formen parte de un bucle y vigile el recálculo del árbol de expansión.

Punto de control 1 (25%): Muestre al profesor el escenario en funcionamiento, así como la adaptación de STP ante fallos en esa red.

6. Creación de los contenedores balanceador y configuración de VRRP

Emplearemos LXC/LXD para la creación de contenedores en VM1 y VM2. Consiga que ambas VMs tengan acceso a Internet a través de su interfaz NAT.

Descargue en ambas VMs mediante *lxc* la imagen de la alpine/3.11. La emplearemos tanto para los balanceadores como para los servidores. En los balanceadores instalaremos *keepalived* para el soporte de VRRP y *haproxy* para actuar como balanceador en capa 7. En los servidores instalaremos Apache con soporte de PHP para actuar como servidor web.

Una vez descargada esta imagen en las dos VMs no necesitaremos acceso a Internet directamente en VM1 y VM2, pero sí lo necesitaremos en los contenedores para poder

instalar software en ellos. Para simplificar esta tarea se le recomienda lo siguiente (en cada una de las VMs 1 y 2):

- Desconfigure la dirección IP que tenga el interfaz NAT (vale con darle la dirección 0)
- Cree un bridge (este bridge no aparece en las figuras anteriores) que llamaremos *br0*
- Asigne el interfaz de NAT a dicho bridge

A partir de aquí daremos acceso a Internet a los contenedores que queramos conectándolos a br0 mediante veths.

Cree un contenedor en VM1 a partir de la imagen que descargó. Este contenedor actuará como el balanceador, al que llamaremos bal1.

Cree un par de veth; uno de ellos asígnelo al namespace de bal1 y el otro conéctelo a *br0*. Dentro de ese contenedor puede emplear un cliente de DHCP para conseguir dirección IP del servidor de DHCP que tiene por defecto el NAT de VirtualBox y con ello tener acceso a Internet dentro del contenedor bal1. En la Alpine tiene el cliente de DHCP *udhcpc*.

Instale en bal1 los paquetes *keepalived, haproxy, curl* y *tcpdump*. El cliente gestor de paquete de Alpine se llama apk (ejemplo: *apk add keepalived*).

El contenedor bal1 no necesita más acceso a Internet. Puede destruir los veth con los que le dio acceso a *br0* o puede parar y volver a arrancar (stop y start) el contenedor, con lo que esos veth deberían desaparecer (el que está en el namespace del contenedor desaparece al terminar éste y el otro con él).

Cree una pareja de veth con los que conectar el contenedor bal1 al puente vswout1. Configure en el contenedor bal1 la dirección IP 172.16.1.250/24.

Para configurar *keepalived* para que gestione mediante VRRP la dirección IP virtual que queremos se debe editar su fichero de configuración /etc/keepalived/keepalived.conf

El contenido básico que necesitamos en este fichero es:

```
vrrp_instance vrrpext {
        state BACKUP
        interface vbal1out_b
        virtual_router_id 12
        priority 150
        advert_int 1
        virtual_ipaddress {
                172.16.1.254
        }
}
```

Las diferentes directivas de configuración son:

- 'vrrp_instance' : Nos permite dar un nombre a esta instancia de VRRP porque el demonio *keepalived* puede tener varias configuradas.
- 'state' : Indica el estado en el que se inicia.
- 'interface' : Configuramos aquí el nombre del interfaz que está conectado a la subred en la que empleamos VRRP. En este caso es el veth que conecta bal1 con vswout1. sustituya 'vbal1out_b' por el nombre que le haya dado a ese interfaz.
- 'virtual_router_id' : Un número de 1 a 255 para distinguir instancias de VRRP en la misma subred (puede seleccionar un valor diferente al 12 del ejemplo).
- 'priority' : Un número de 1 a 255 tal que el interfaz configurado con mayor valor será el maestro (en caso de empate se desempata por la dirección IP del interfaz).

- 'advert_int' : Se indica el número de segundos entre los envíos de mensajes VRRP que hace el maestro para que los equipos de backup del grupo sepan que sigue vivo.
- 'virtual_ipaddress' : Permite indicar la dirección IP virtual (puede ser más de una).

Para más detalle recurra a la documentación de *keepalived*[1]. Este software puede actuar también como un balanceador en capa 4 pero no vamos a emplear esta funcionalidad.[2]

Lance el demonio *keepalived* ejecutando en el contenedor bal1:

```
/etc/init.d/keepalived start
```

Repita los pasos descritos para crear el balanceador bal2 que corre en VM2. Su dirección IP en la subred externa será 172.16.1.251/24. Configure un valor menor de prioridad en VRRP para este contenedor para que actúe como backup.

En este momento debería tener funcionando lo que se representa en la Figura 7. Debería tener conectividad IP entre 'usuario' y las direcciones externas de los dos balanceadores, así como con la dirección IP virtual que protegen entre los dos.

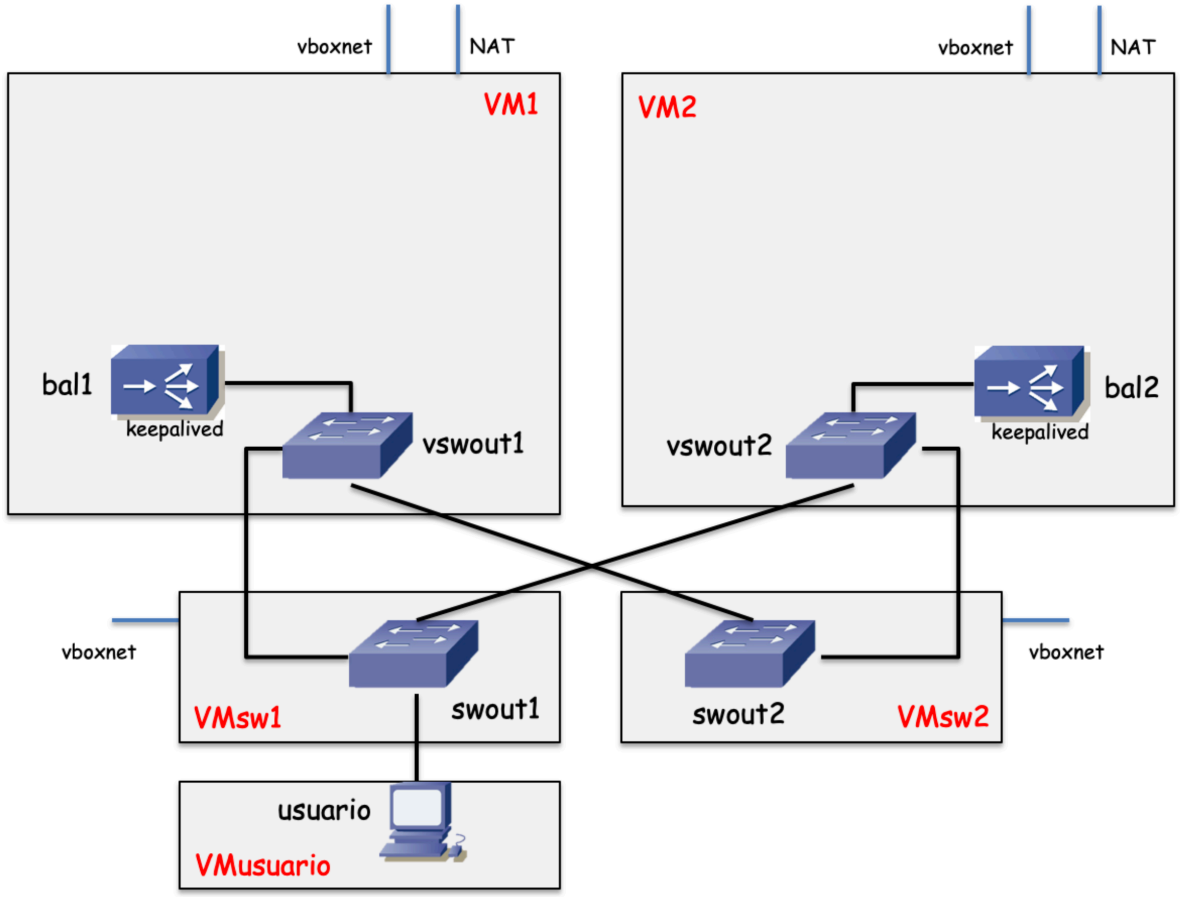

Figura 7 - Escenario tras la configuración del interfaz externo de los balanceadores

Puede ver los cambios de estado de VRRP en el fichero /var/log/messages de cada contenedor balanceador. Pruebe por ejemplo a desactivar (*ifconfig down*) el puente vswout1 con lo que el maestro VRRP pierde conectividad y vea si toma el rol de maestro el contenedor bal2.

---

[1] https://keepalived.readthedocs.io/en/latest/configuration_synopsis.html#vrrp-instance-definitions-synopsis

[2] Con esta configuración no se está empleando la dirección MAC reservada por IANA para los interfaces virtuales sino que contestan al ARP que pregunta por la IP virtual directamente con la dirección MAC de su NIC.

Punto de control 2 (30%): Muestre al profesor el escenario en funcionamiento, así como la adaptación empleando VRRP ante fallos en esa red.

7. Creación de los servidores

Lance en VM1 un contenedor a partir de la imagen alpine/3.11 que servirá como uno de los servidores. Lo llamaremos serv1.

Dé acceso a Internet a este contenedor de la misma forma que lo hizo para el contenedor balanceador, ya que necesitamos instalar software en él.

Instale en él los paquetes *apache2, php-apache2* y *tcpdump.*

Detenga ese contenedor para que pierda el interfaz que le daba salida a Internet. Además cree un snapshot del contenedor mediante:

```
lxc snapshot serv1 conApache
```

Ahora crearemos una nueva imagen a partir de esa snapshot mediante:

```
lxc publish ser1/conApache --alias alpineApache
```

A partir de esta nueva imagen (de nombre *alpineApache*) podemos crear más contenedores en VM1 y ya tendrán el software anterior instalado.

Cree en VM1 el puente vswin1.

Arranque el contenedor serv1. Cree un par de veth conectando un extremo a vswin1 y el otro asignándolo al namespace de serv1. Configure al interfaz del contenedor serv1 la dirección IP 10.0.0.1/24.

El servidor Apache instalado en serv1 ofrece por defecto los documentos que tiene en /var/www/localhost/htdocs/. Cree ahí por ejemplo un fichero index.php con un PHP que le permita identificar en la página devuelta qué dirección IP tiene el cliente que está accediendo al servidor. Por ejemplo en serv1 podríamos poner este index.php:

```
<html><body><h1>Esto es el servidor serv1</h1>
<p>El cliente que ha pedido esta pagina tiene direccion:
<?php
print_r(\$_SERVER['REMOTE_ADDR']);
?>
</body></html>
```

Lance el servidor Apache en serv1 mediante:

```
/etc/init.d/apache2 start
```

Cree un par de veth conectando un extremo a vswin1 y el otro asignándolo al namespace de bal1. Configure al interfaz del contenedor bal1 la dirección IP 10.0.0.250/24.

Compruebe la conectividad IP entre serv1 y bal1 a través de la subred interna.

Lance un segundo contenedor de nombre por ejemplo serv2 a partir de la imagen *alpineApache*. Repita los pasos para darle acceso a la subred interna y configure en él la dirección 10.0.0.2/24. Cree en él un nuevo index.php que permita identificarlo y lance su servidor web.

Repita los pasos en VM2 para tener allí un contenedor serv3 con el software Apache conectado a un puente vswin2 al cual esté conectado también el balanceador bal2. El balanceador bal2 tendrá dirección IP interna 10.0.0.251/24. El contenedor serv3 tendrá dirección 10.0.0.3/24. Puede repetir todos los pasos para crear la imagen o probar a controlar el demonio *lxd* de VM1 desde el cliente *lxc* en VM2 para lanzar en VM2 el contenedor serv3 a partir de la imagen que ya tiene creada en VM1.

En estos momentos debería tener algo similar a lo que se muestra en la Figura 8, a falta tan solo de configurar y lanzar el software de balanceador *haproxy* en bal1 y bal2. Fíjese en que tal y como está el montaje no hay conectividad entre los contenedores servidor de VM1 y los de VM2 ni bal1 puede comunicarse por la subred interna con serv3 ni bal2 puede con serv1 y serv2. La subred interna está partida, no hay conectividad en capa 2 entre vswin1 y vswin2. Esto lo arreglaremos en la próxima sección de la práctica, pero antes dejaremos configurados los balanceadores.

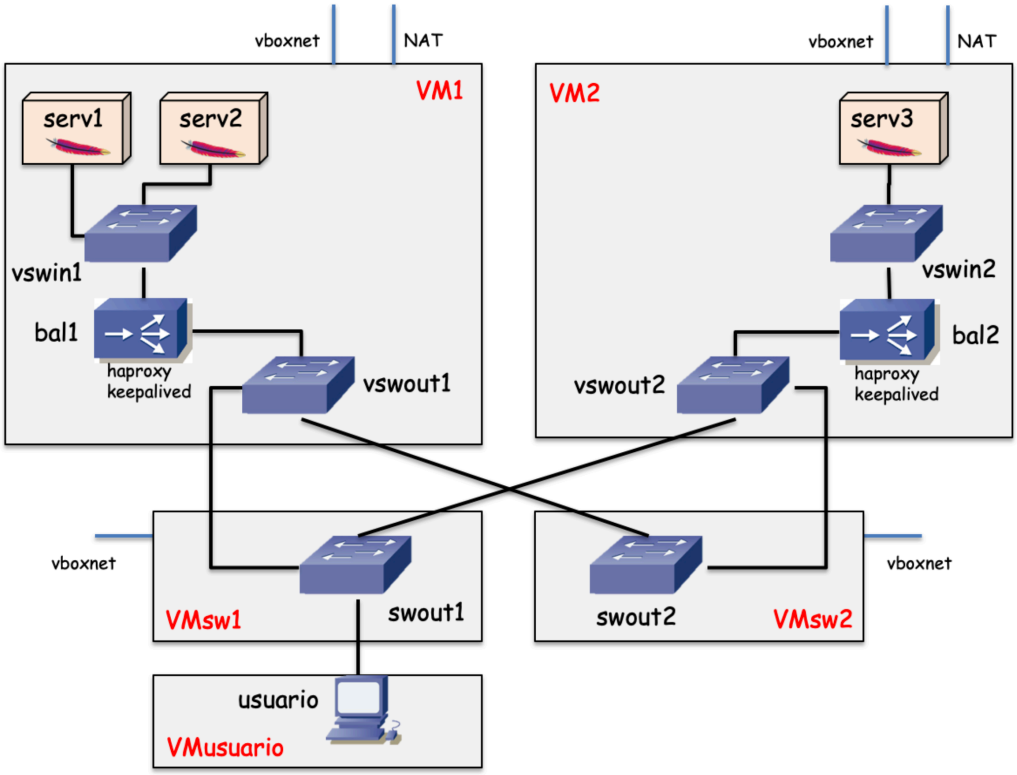

Figura 8 - Escenario con todos los contenedores creados

Para configurar *haproxy* en bal1 y en bal2 deberá añadir al menos un conjunto similar a las directivas siguientes al fichero /etc/haproxy/haproxy.cfg :

```
frontend miservidor
  bind *:8000
  default_backend misServidores
backend misServidores
  balance roundrobin
  server serv1 10.0.0.1:80 check
  server serv2 10.0.0.2:80 check
  server serv3 10.0.0.3:80 check
```

Es sencillo intuir de esta configuración [3] que el balanceador aceptará conexiones a cualquiera de sus interfaces al puerto 8000 y que las peticiones HTTP que reciba las va a repartir mediante un algoritmo *round robin* entre los servidores de direcciones 10.0.0.1, 10.0.0.2 y 10.0.0.3, haciendo conexiones a sus puertos 80 en todos ellos (donde hemos dejado el servidor web Apache). La opción 'check' activa que el balanceador compruebe periódicamente que dichos servidores siguen funcionando, para retirarlos de uso en caso de que pierda conectividad con ellos. Ahora mismo, si ponemos esta configuración en bal1 repartirá peticiones solo entre serv1 y serv2 pues al no tener conectividad con serv3 le

[3] https://www.haproxy.com/blog/the-four-essential-sections-of-an-haproxy-configuration/

fallan sus comprobaciones con él y lo retirará de la lista de servidores entre los que las reparte, así que no le mandará peticiones. Lo complementario sucedería desde bal2.

Relance *haproxy* en los balanceadores tras cambiar su configuración:

```
/etc/init.d/haproxy restart
```

Compruebe que puede ir con un navegador en la VMusuario a http://172.16.1.254:8000. Debido a que bal1 es el maestro del grupo VRRP debería dirigirse el tráfico hacia bal1 y éste balancear sus peticiones entre serv1 y serv2, con lo que verá que peticiones consecutivas van alternándose entre ser atendidas por uno u otro servidor. Lo puede ver en el tráfico y en la respuesta que da el script PHP. También verá en la respuesta del script PHP que la petición que ha recibido el servidor le ha llegado de la dirección IP interna de un balanceador (el cual está actuando como proxy).

Punto de control 3 (30%): Muestre al profesor el escenario en funcionamiento (con bal1 atendiendo a las peticiones).

Cree algún problema de conectividad en la subred externa tal que las peticiones acaben en el segundo balanceador porque pase él a ser el maestro del grupo VRRP. Con ello podrá ver cómo todas las peticiones son atendidas ahora por serv3.

8. Extensión de la subred interna

Para completar el escenario propuesto nos falta solo que todos los servidores se encuentren en la misma subred IP, en el mismo dominio de broadcast. Para lograr esto podemos enlazar vswin1 con vswin2 empleando VXLAN. La comunicación VXLAN será entre las direcciones IP de VM1 y VM2. Consulte prácticas anteriores sobre cómo llevar a cabo esta configuración.

Compruebe ahora que en la subred interna hay comunicación correcta entre los 5 interfaces IP (los 3 servidores y los 2 balanceadores) y que en funcionamiento normal las peticiones web dirigidas a bal1 se reparten entre los 3 servidores. Podrá ver el tráfico VXLAN entre VM1 y VM2 cuando bal1 envía una petición a serv3.

Punto de control 4 (15%): Muestre al profesor el escenario en funcionamiento.

9. Conclusión

Consulte la documentación de *haproxy* para ver otras opciones que tiene como por ejemplo para diferentes comprobaciones (*health tracking*), para inserción de *cookies*, para repartir las peticiones entre diferentes conjuntos de servidores en función del tipo de documento solicitado, diferentes algoritmos de reparto por ejemplo con pesos, etc.

Podríamos añadir más servidores creando más contenedores en estos hosts, sin embargo, seguramente es mucho más útil extender la VXLAN a más hosts donde correr más contenedores servidor y repartir la carga entre muchos más contenedores servidor que están en diferente hardware, con lo que añadimos más CPUs. ¿Sería necesario un balanceador en cada uno de esos nuevos hosts?

Toda la configuración llevada a cabo en esta práctica da un ejemplo del tipo de tareas que llevan a cabo aplicaciones distribuidas como Kubernetes, OpenStack o Docker Swarm.