1. Introducción

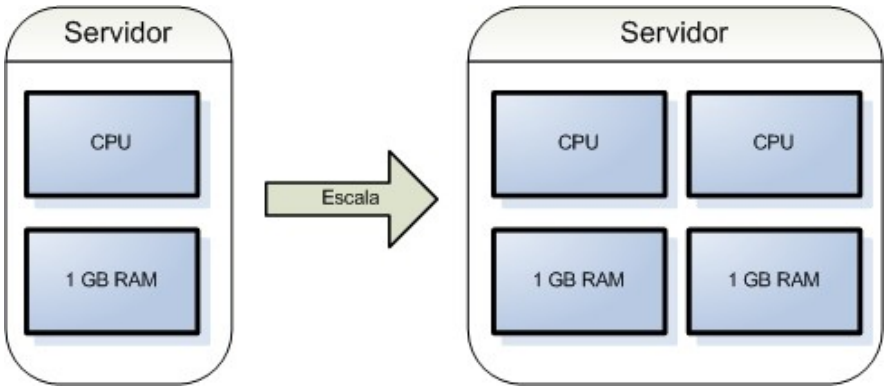
A la hora de escoger una arquitectura para un proyecto se deben tener en cuenta diversos factores como el rendimiento, escalabilidad, disponibilidad, seguridad, coste y facilidad de gestión.

2. Escalabilidad

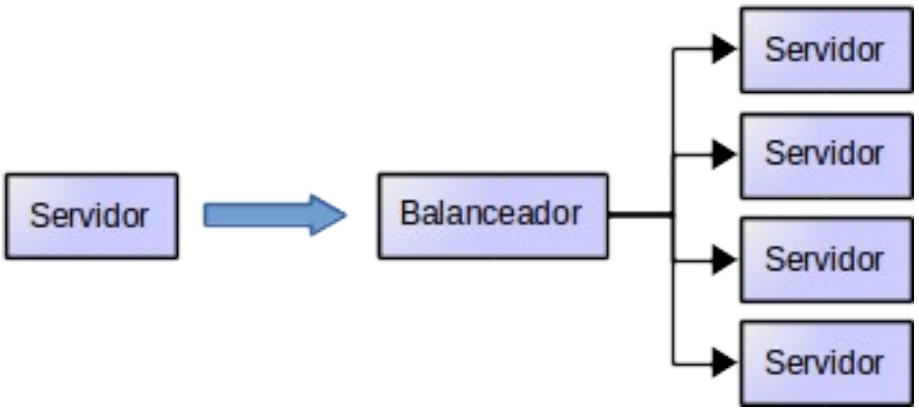
La escalabilidad de un sistema informático indica su capacidad para crecer sin perder calidad en los servicios ofrecidos. Es decir, la suficiencia de dicho sistema informático de variar su tamaño, características y capacidad de servicios para adaptarse a una nueva situación.

Dicho de modo resumido, es la capacidad de un sistema informático de adaptarse a una nueva situación sin que el servicio que ofrece se vea afectado. Existen dos tipos de escalabilidad:

- **Escalabilidad vertical**: partiendo de un sistema informático con unos recursos determinados, se añaden más recursos a dicho sistema (memoria, disco duro, procesador, …).



- **Escalabilidad horizontal**: Se añaden más servidores con la misma funcionalidad al sistema, redistribuyendo la carga entre todos ellos.



El principal problema de escalar un sistema informático es estimar cuánta carga deberá soportar el sistema. Proporcionar una cifra exacta es complicado y en el momento del diseño se deben realizar estimaciones. Cuando el sistema esté en producción se debe monitorizar para ver si su dimensionamiento es el adecuado o no. Si la cifra se sobrestima, se desperdiciarán recursos y si se subestima, el sistema no estará preparado para soportar el aumento de carga.

A la hora de pensar en el escalado vertical hay que tener en cuenta dos factores:

- Límite del hardware: por ejemplo, si una placa base soporta hasta 64 GB de RAM no se pueden aumentar la RAM por encima de ese valor.

- Relación coste-rendimiento: aumentar las prestaciones de un servidor tiene un precio, y se llega a un momento en que es más barato tener dos servidores que uno sólo que ofrezca el mismo rendimiento.

A la hora de pensar en el escalado horizontal hay que tener en cuenta los siguientes factores:

- Mayor complejidad de gestión: por un nº superior de más nodos a administrar.
- Posibles problemas de latencia y complejidad al diseño aplicaciones.

A día de hoy y gracias a los avances en la virtualización hay una clara tendencia al escalado horizontal.

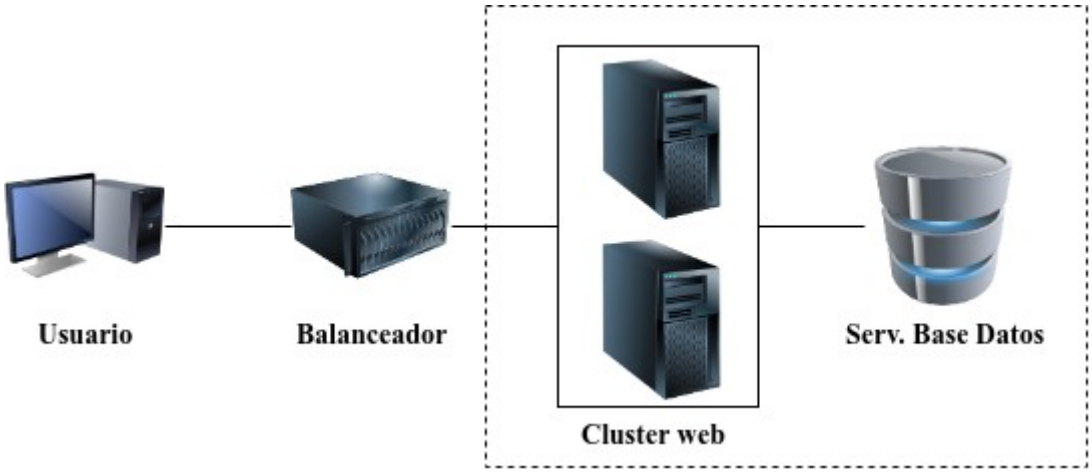
3. Redundancia y tolerancia a fallos

Los sistemas redundantes son aquellos en los que se duplican algunos componentes de carácter crítico. El objetivo de repetir estos dispositivos críticos es que el sistema siga proporcionando servicio aunque se produzca un error en alguno de estos. Si se produce un fallo en alguno de los componentes, el sistema seguirá proporcionando servicio aunque, seguramente, con un rendimiento menor.

Para proporcionar una tolerancia a fallos, un sistema debe cumplir una serie de características:

- No debe existir un único punto de fallo[1]: la redundancia de componentes debe asegurar la continuidad del servicio ofrecido. Además, se debe proporcionar métodos para la reparación sin afectar al servicio.
- Aislamiento del fallo: si se produce algún error, se debe identificar y aislar el componente que lo ha causado sin que afecte a otros..
- Vuelta atrás: debe existir algún método para volver a una situación previa en la cual el sistema funcionase correctamente.

En el siguiente sistema tanto el balanceador (*load balancer*) como el *database server* son SPOF. Si uno de los dos cayese, el sistema dejaría de dar servicio a los usuarios.



En la escalabilidad vertical no tiene porqué estar implícita una tolerancia a fallos del sistema. El que un sistema tenga dos procesadores o dos módulos de memoria no quiere decir que el sistema pueda seguir ofreciendo servicio si se produce un error en alguno de ellos.

En cambio, la tolerancia a fallos viene implícita cuando se habla de escalabilidad horizontal. Se cuenta con un balanceador de tráfico que es el encargado de monitorizar el estado de cada uno de

[1]Single Point Of Failure o SPOF ("punto único de fallo") es un componente de un sistema que tras un fallo en su funcionamiento ocasiona un fallo global en el sistema completo, dejándolo inoperante. Un SPOF puede ser un componente de hardware, software o eléctrico. Para garantizar la inexistencia de un SPOF en un sistema, debe haber redundancia, que garantiza el correcto funcionamiento aún en caso de que alguno de sus componentes falle.

los servidores y decidir si está operativo o no. En este caso, el balanceador de tráfico se convierte en el único punto de fallo, ya que si este falla, todo el servicio fallará.

4. Alta disponibilidad (High Availability)

Con la alta disponibilidad se busca que un servicio siempre esté disponible para los usuarios, idealmente, 24 horas al día, los 7 días de la semana y los 365 días del año. Para lograr la alta disponibilidad se tiene que recurrir a la redundancia tanto a nivel de software como de hardware. De esta forma, se pueden abordar tanto situaciones de fallos inesperados (p.e. fallos en discos duros, avería de un switch, caída de una línea WAN, …) como de paradas programadas (p.e. actualización del hardware de algún equipo, actualizaciones de SO y aplicaciones, …). En el ejemplo del punto anterior se debería añadir otro *load balancer* y otro *database server* para lograr la alta disponibilidad.

La disponibilidad se suele expresar como un porcentaje del tiempo de funcionamiento en un año dado: *(total time - down time)/total time*. En la siguiente tabla se pueden observar la traducción del porcentaje de disponibilidad a cantidad de tiempo de caída/inactividad de un sistema:

Availability %	Downtime per year	Downtime per month	Downtime per week
90% ("one nine")	36.5 days	72 hours	16.8 hours
95%	18.25 days	36 hours	8.4 hours
97%	10.96 days	21.6 hours	5.04 hours
98%	7.30 days	14.4 hours	3.36 hours
99% ("two nines")	3.65 days	7.20 hours	1.68 hours
99.5%	1.83 days	3.60 hours	50.4 minutes
99.8%	17.52 hours	86.23 minutes	20.16 minutes
99.9% ("three nines")	8.76 hours	43.8 minutes	10.1 minutes
99.95%	4.38 hours	21.56 minutes	5.04 minutes
99.99% ("four nines")	52.56 minutes	4.32 minutes	1.01 minutes
99.995%	26.28 minutes	2.16 minutes	30.24 seconds
99.999% ("five nines")	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% ("six nines")	31.5 seconds	2.59 seconds	604.8 milliseconds
99.99999% ("seven nines")	3.15 seconds	262.97 milliseconds	60.48 milliseconds
99.999999% ("eight nines")	315.569 milliseconds	26.297 milliseconds	6.048 milliseconds
99.9999999% ("nine nines")	31.5569 milliseconds	2.6297 milliseconds	0.6048 milliseconds

Generalmente los SLA (Service Level Agreements) hacen referencia a valores de disponibilidad o downtime mensuales para calcular las devoluciones a efectuar a los clientes por fallos en los sistemas. A modo de ejemplo, se muestra una parte del SLA de la empresa gallega de hosting Dinahosting:

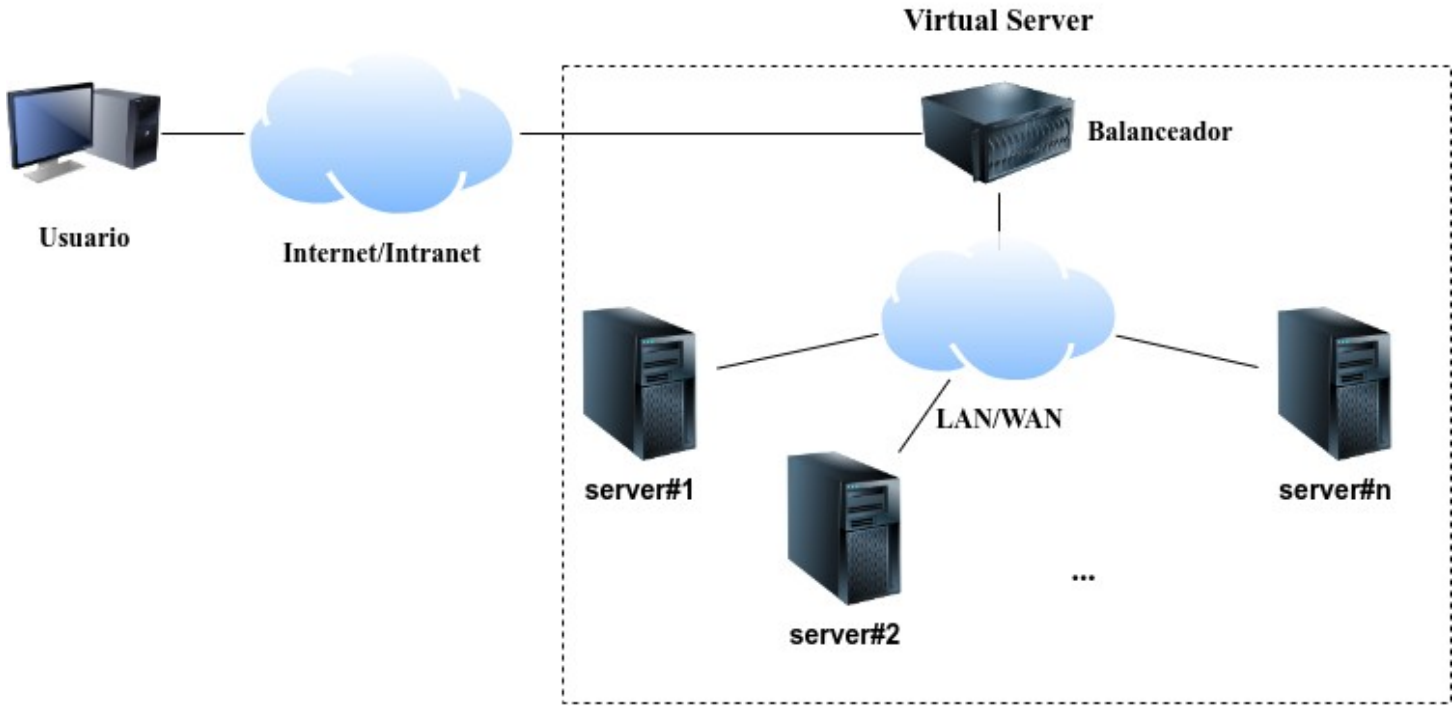
Incidencias de red: DINAHOSTING garantiza una disponibilidad de Red del Servicio del 99,9% %. En caso de indisponibilidad real de este nivel de servicio, se aplicarán las penalizaciones siguientes:

- *<99,9%% de disponibilidad mensual → 5% devolución de importe mensual*
- *<99% de disponibilidad mensual → 25% devolución de importe mensual*
- *<98% de disponibilidad mensual → 50% devolución de importe mensual*
- *<96% de disponibilidad mensual → 75% devolución de importe mensual*

- *<90% de disponibilidad mensual → 100% devolución de importe mensual*

5. Balanceadores de carga

Los balanceadores de carga permiten introducir mejoras en el rendimiento y en la disponibilidad del sistema, al distribuir la carga de trabajo entre varios servidores. Si uno de los servidores balanceados falla, el balanceador redirigirá las peticiones a los otros servidores hasta que el servidor caído se recupere. El balanceo se puede hacer a nivel 4 (L4 o nivel de transporte) o a nivel 7 (L7 o nivel de aplicación)[2]. Ejemplo de software capaz de hacer esta función sería: LVS (Linux Virtual Server), HAProxy, Apache, Nginx, Varnish.



Cuando se trabaja con balanceadores de carga aparece el concepto de *Virtual Server* (VS) que se corresponde con un servidor altamente disponible y altamente escalable construido en base a un cluster de servidores reales. La arquitectura del cluster es completamente transparente para los usuarios finales, ya que éstos interactúan con el cluster como si se tratase de un único servidor de alto rendimiento. Los balanceadores de carga y los servidores reales pueden estar conectados tanto por una LAN de alta velocidad como estar dispersos geográficamente.

Ventajas:

- Permite escalar horizontalmente añadiendo más servidores.
- Protección frente DDoS al poder limitar las conexiones por cliente (cantidad y frecuencia) y distribuir la carga de trabajo entre varios servidores de backend.

Desventajas:

- El balanceador de carga puede convertirse en un cuello de botella si no tiene suficientes recursos o está incorrectamente configurado.
- En función de la aplicación pueden surgir complicaciones como sesiones persistentes[3] o terminación SSL.

6. Algoritmos de balanceo

[2] Si trabajamos a nivel 7, el encargado del balanceo es un proxy inverso (*reverse proxy*).

[3] Sesiones persistentes o sticky sessions: una vez que un cliente haya establecido una conexión con un servidor backend, en sucesivas visitas el cliente vuelva a trabajar con ese mismo servidor.

La decisión de a qué servidor debe reenviar la petición del usuario se realiza mediante un conjunto de algoritmos de planificación. El principal objetivo de un algoritmo de balanceo es decidir a qué servidor en concreto debe reenviar una petición. Esta decisión dependerá de una serie de factores determinados por el algoritmo de balanceo elegido. A continuación se realiza una breve descripción de los algoritmos de balanceo más populares soportados por balanceadores como HAProxy y LVS.

- **Round Robin:** distribución equitativa entre los servidores disponibles. Se trata de una buena opción si el tráfico a repartir entre los servidores es similar; es decir, se espera que las solicitudes se procesen todas en un tiempo parecido. En el caso de no ser así; por ejemplo consultas a bases de datos, podría ocurrir que un servidor se vea sobrecargado al seguir recibiendo peticiones mientras está procesando consultas especialmente largas.

  En el siguiente código se puede ver como hacer roundrobin con HAProxy:

```
frontend web
        bind *:80
        use_backend cluster_web
backend cluster_web
        balance roundrobin
        server web01 192.168.10.130:80
        server web02 192.168.10.131:80
        server web03 192.168.10.132:80
        server web04 192.168.10.133:80
```

- **Weighted Round Robin:** asigna las peticiones a los servidores web según un peso asignado previamente. Los servidores con mayor peso (p.e con mejores prestaciones hardware) reciben nuevas peticiones antes y, por tanto, procesan más peticiones que los servidores con menor peso. A igualdad de peso, la distribución se realiza de forma equitativa.

```
frontend web
        bind *:80
        use_backend cluster_web
backend cluster_web
        balance roundrobin
        server web01 192.168.10.130:80 weight 3
        server web02 192.168.10.131:80 weight 2
        server web03 192.168.10.132:80 weight 1
        server web04 192.168.10.133:80 weight 1
```

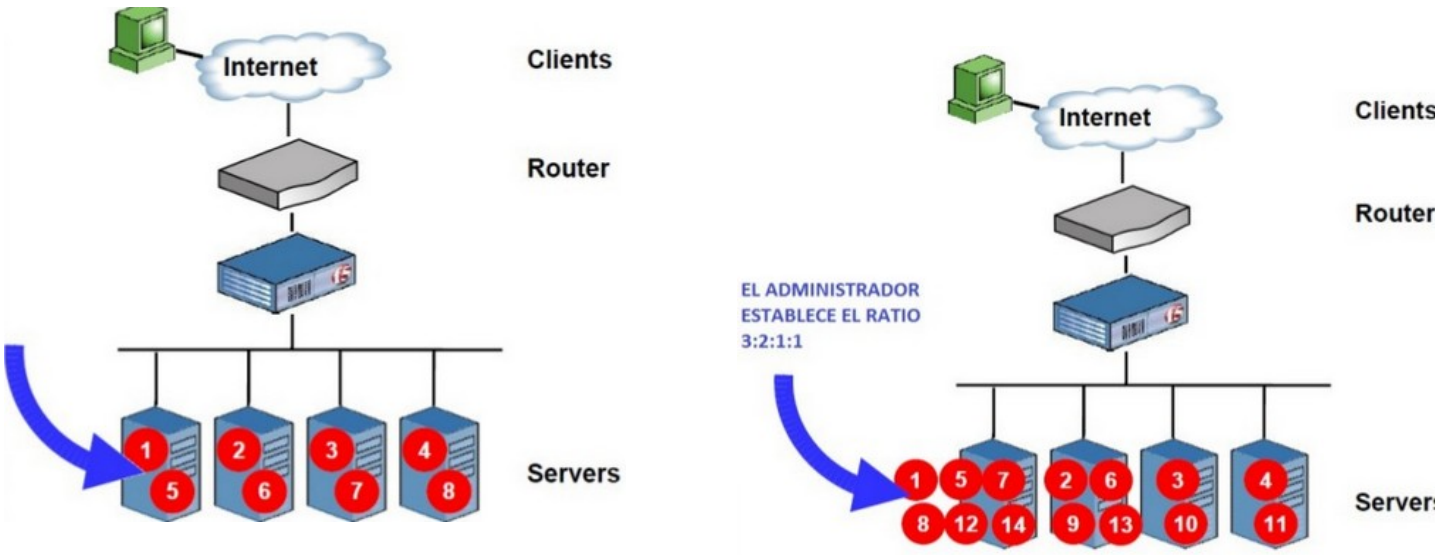


Fig. Round Robin y Weighted Round Robin

- **Least Connection:** el balanceador envía las nuevas peticiones a los servidores con el menor número de peticiones activas. Se trata de soluciones los posibles problemas que aparecen con RoundRobin en el caso de solicitudes que implican un tiempo de resolución alto.

  En el siguiente código se puede ver como configurar *least connection* para dos bases de datos en HAProxy:

```
frontend mysql
        mode tcp
        bind *:3303
        default_backend cluster_mysql
backend cluster_mysql
        mode tcp
        balance leastconn
        server db01 192.168.10.115:3303
        server db02 192.168.10.116:3303
        server db03 192.168.10.117:3303
        server db04 192.168.10.118:3303
```

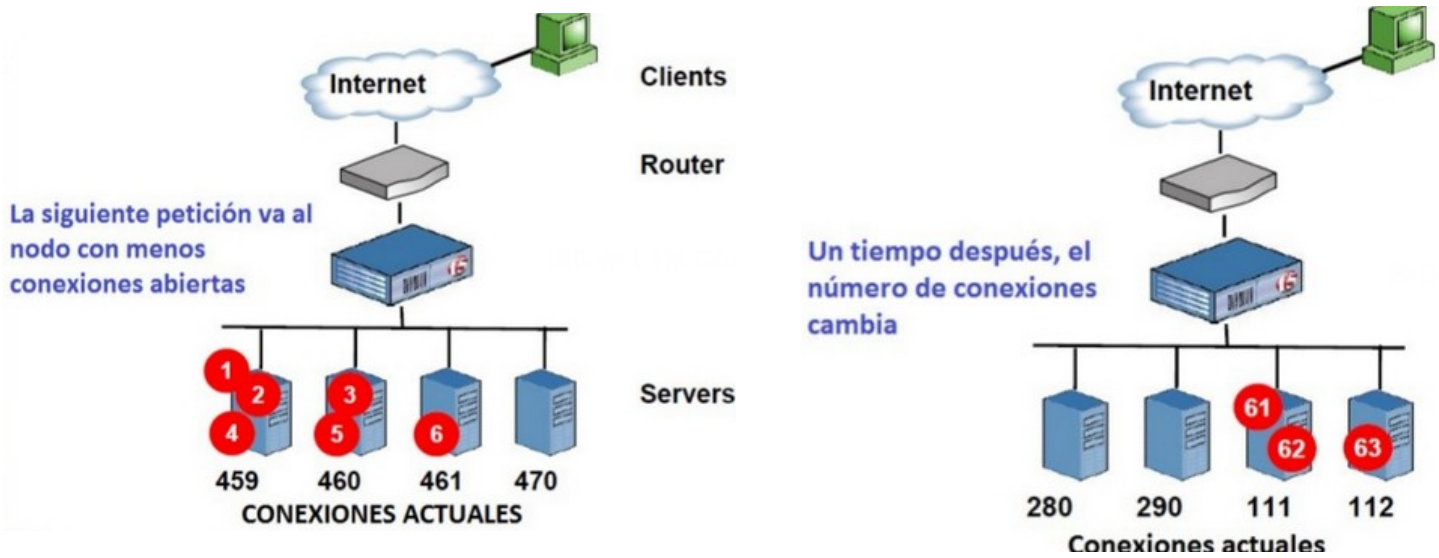


Fig. Least Connection

- **Weighted Least Connection:** *least connection* procura dividir el tráfico de forma que los servidores más ocupados reciban menos solicitudes. Para ello mira únicamente las conexiones que atienden los servidores sin valor nada más. Con *weighted least connection* se asigna un peso a cada servidor de forma que es posible valorar otras características como cpu y/o ram. Asigna más peticiones a los servidores con menores peticiones activas en función del peso específico de dicho servidor: *Ci / Wi*. Donde *Ci* es el número de peticiones activas y *Wi* el peso específico del servidor.

```
frontend mysql
        mode tcp
        bind *:3303
        default_backend cluster_mysql
backend database_replicas
mode tcp
        balance leastconn
        server db01 192.168.10.115:3303 weight 4
        server db02 192.168.10.116:1433 weight 1
```

- **Hash URI algorithm:** se analiza el path de la URL solicitada y reenvía solicitudes idénticas hacia el mismo servidor de backend. Por ejemplo, las peticiones a la URL `http://www.manuel.com/logo.png` irían al mismo servidor interno. Para hacer esto, HAProxy almacena un hash de la URL junto con el identificador del servidor al que se le envió la solicitud la última vez. Este algoritmo está especialmente pensado para combinarlo con sistemas de cachés colocados antes de los servidores webs reales y poder aprovechar al máximo los hits.

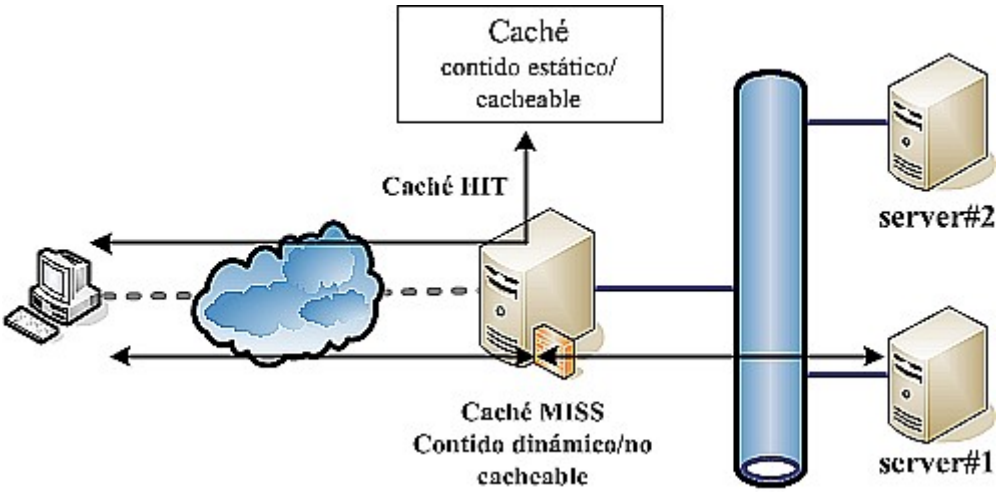
```
backend cache_servers
      balance uri
      server cache01 192.168.10.200:80
      server cache02 192.168.10.201:80
```

Cuando todos los servidores reales fallan, es posible configurar un servidor de backup o **Fallback** al que se le envían las peticiones mientras no vuelvan a estar operativos los servidores.

7. Proxy Inverso (Reverse Proxy)

El gran beneficio de los tipos de proxy estándar o caché vistos en el tema anterior, radica en que se utiliza de forma más eficiente la conexión a Internet (cuando se trata de protocolos cacheables); dando la impresión a los clientes, de estar trabajando a una velocidad superior. En cambio, un proxy inverso trata de liberar parte de la carga de los servidores ubicados tras él, como se explica en el siguiente ejemplo:

Cuando un cliente web (navegador) realiza una consulta http, el sistema dns lo encamina hacia el proxy inverso, no hacia el servidor real. Cuando la solicitud del cliente llega al proxy inverso, éste comprueba su caché para ver si contiene el elemento solicitado. En caso de no tenerlo, contactará con el servidor web real y descargará el elemento a su caché y lo enviará al cliente. El proxy inverso únicamente puede guardar URLs cacheables (como páginas html e imágenes). El contenido dinámico, como páginas PHP no puede ser cacheado.



Existen cuatro cabeceras http en relación a la caché del servidor proxy:

- `Last-Modified`: permite saber al proxy cuando fue la última modificación del recurso.
- `Expires`: le indica al proxy cuando debe eliminar el recurso de la caché.
- `Cache-Control`: le indica al proxy si el recurso debe ser cacheado.
- `Pragma`: también sirve para indicarle al proxy si el recurso debe ser cacheado.

El proxy inverso recibe las peticiones y las reenvía a los servidores web. La ubicación del proxy inverso hace que todo el tráfico entrante desde Internet con destino a esos servidores pase por él, lo que conlleva ciertas ventajas:

- Seguridad: el servidor proxy es una capa adicional de defensa ya que en ningún momento se accede directamente a los servidores web sino al proxy.
- Balanceo de Carga: el proxy inverso puede balancear la carga entre varios servidores enviando las peticiones a aquellos servidores que se encuentren más descargados.
- Re-escritura y validación de las peticiones: el proxy inverso puede necesitar reescribir las URL de cada página web (traducción de la URL externa a la URL interna correspondiente, según en qué servidor se encuentre la información solicitada).
- Caché de contenido estático: un proxy inverso puede ayudar a descargar a los servidores web almacenando contenido estático como imágenes, documentos, ficheros JavaScript, hojas de estilo CSS u otro contenido gráfico.
- Cifrado / Aceleración SSL: en caso de sitios webs seguros, habitualmente el cifrado SSL no lo hace el mismo servidor web, sino que es realizado por el proxy inverso, el cual está equipado con un hardware de aceleración SSL.

Debido a su funcionalidad, a un proxy trabajando como proxy inverso también se le conoce como *Server Accelerator Mode* o *Server Surrogate*. Ejemplos de proxys inversos son squid, varnish, pound y haproxy. Además, los servidores web Apache, nginx y Lighttpd también pueden funcionar como proxys inversos.

HAProxy

HAProxy es un balanceador tcp y http muy rápido y confiable que soporta grandes cantidades de tráfico; y por esta razón se está a usar en un gran número de sitios webs, entre los que están los más visitados de Internet (Instagram, Twitter, Alibaba, etc.). Centrándonos en los aspectos fundamentales hay que decir que HAProxy puede trabajar como balanceador de tipo L4 o L7:

- Balanceador L4: estamos ante un balanceo a nivel de transporte (TCP), similar al que proporciona LVS (Linux Virtual Server) en el kernel de Linux. Centrándonos en sitios web, este tipo de balanceo está pensado para usar con servidores que tengan el mismo contenido, la misma aplicación.
- Balanceador L7: estamos ante un balanceo a nivel de aplicación. Trabajar a nivel de aplicación abre nuevas posibilidades al poder acceder a la capa más próxima al usuario. Centrándonos en la web, podremos tomar decisiones en función de las cabeceras http de los paquetes, insertar cookies, marcar paquetes, ...; y por lo tanto, podremos reenviar las solicitudes de los clientes a diferentes servidores en base al contenido de la petición del cliente. Uno de los usos más interesantes es la posibilidad de correr diferentes aplicaciones en diferentes servidores reales pero accesibles bajo el mismo nombre de dominio y puerto.

En las siguientes imágenes se ve un ejemplo de balanceador L4 para dar servicio a dos servidores con el mismo contenido y un balanceador L7 que da servicio a dos aplicaciones diferentes bajo el mismo nombre de dominio y puerto (si el cliente solicita un recurso bajo /foro su petición se dirige a los servidores del cluster foro, en otro caso va al cluster web):

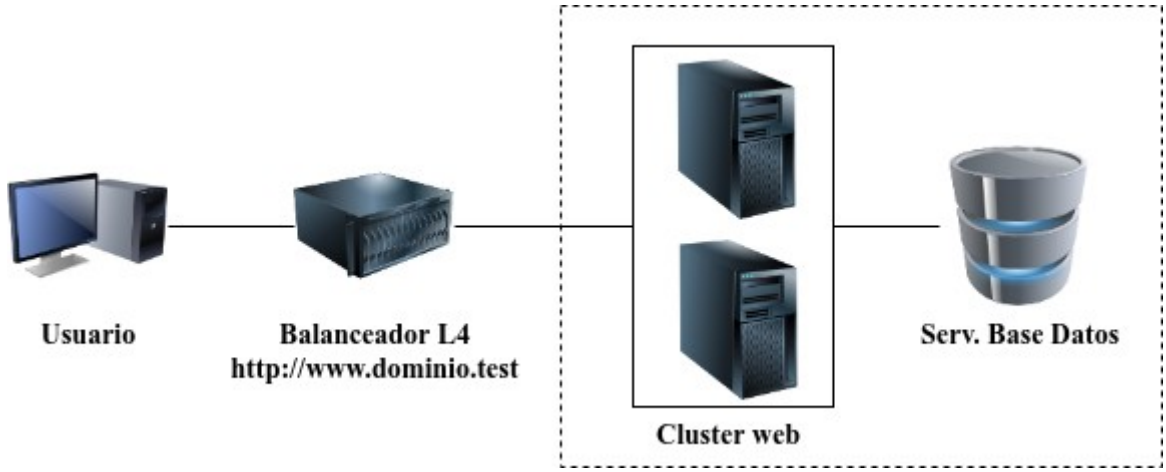


Fig. HAProxy como balanceador L4.

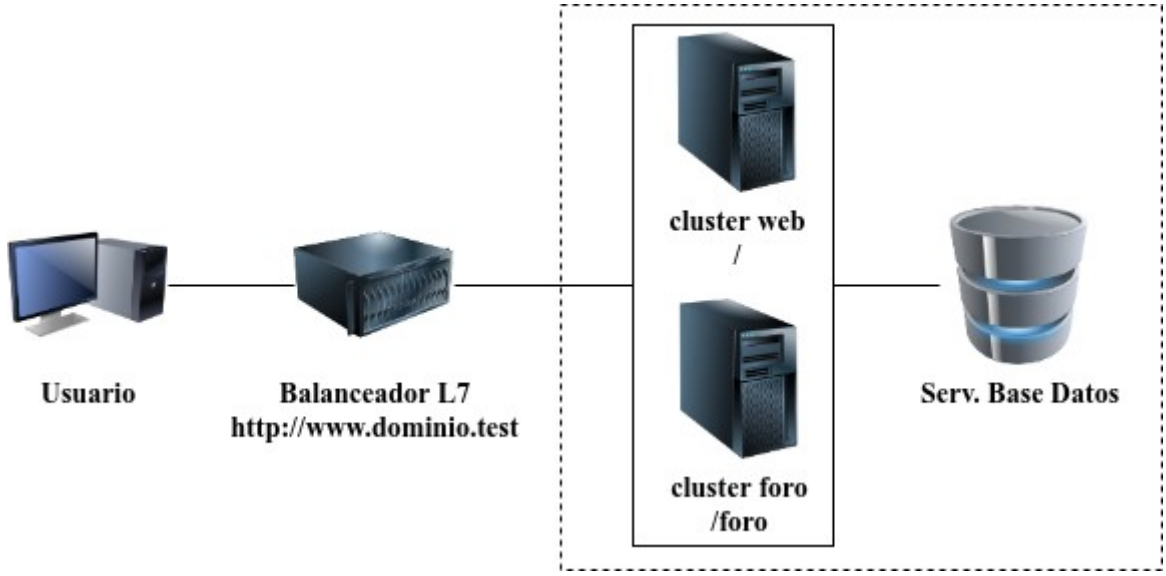


Fig. HAProxy como balanceador L7.

Uno de los usos habituales de HAProxy es el de terminador SSL/TLS. Los clientes que precisen acceder a un servicio web usando comunicaciones https atacan a HAProxy, y éste procede a descifrar la información y contactar con los servidores reales por http. Se trata de ahorrar trabajo a los servidores internos, que son los que realmente corren la aplicación web. HAProxy soporta SNI (*Server Name Indication*) por lo que podemos trabajar con diferentes certificados para dar servicio a diferentes dominios. La siguiente imagen proporciona una visión de este concepto:

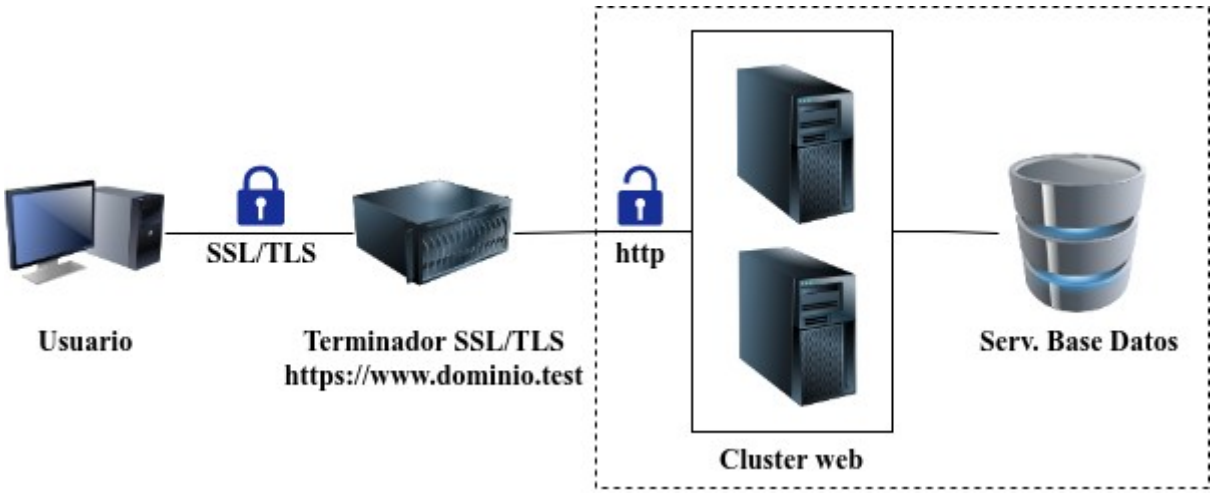
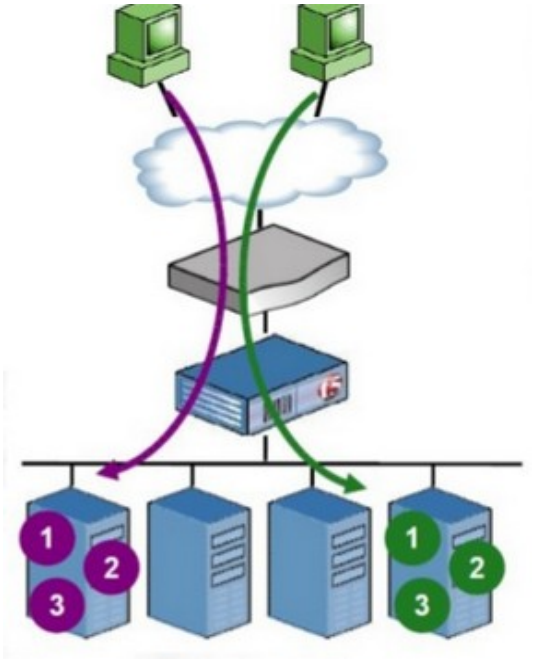


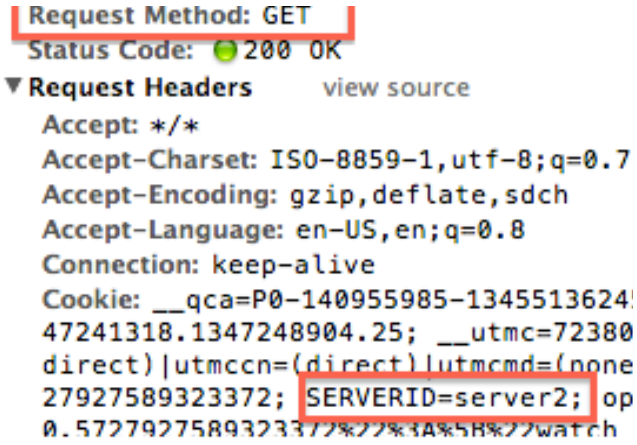
Fig. HAProxy como terminador SSL/TLS.

8. Persistencia de sesiones



El protocolo Http es un protocolo sin estado por lo que a la hora de trabajar con balanceadores de tráfico surge el problema de cómo manejar la información que se debe almacenar durante las múltiples peticiones que se llevan a cabo durante una sesión de un usuario. Si esta información se almacena en los servidores entonces el balanceador debería ser capaz de reenviar las peticiones de ese usuario al mismo servidor cada vez. Si no se hiciese así, la información de sesión podría encontrarse en un servidor diferente al cual ha sido dirigido el cliente. Existen varias soluciones a este problema:

- **Compartir la información de sesión:** almacenando los datos en una base de datos o sistema de ficheros. Esta solución, si bien elimina el problema de la disponibilidad de la información, introduce otras cuestiones como el aumento de carga significativo en la base de datos, latencias derivadas de sistemas de ficheros en red, necesidad de replicación para prevenir que estos servidores se conviertan en puntos únicos de fallo.
- **Afinidad (affinity)**: Se usa información de un nivel inferior a la capa de aplicación para mantener asociado a un cliente con un servidor real.
- **Persistencia (*persistence* o *stickiness)***: Se usa información de nivel de aplicación para mantener asociado a un cliente con el servidor real. Existen dos técnicas principales:
  1. ***Cookies*:** una *cookie* no es más que información que se almacena en el ordenador del visitante de una página web. El balanceador introduce una cookie en la respuesta del servidor al usuario o modifica una cookie enviada por el servidor al cliente; de forma que, futuros paquetes desde ese equipo cliente tendrán la cookie y el balanceador podrá enviar el paquete el servidor real adecuado.



  2. ***URL Rewriting*:** consiste en ir modificando la *URL* introduciendo en ella los datos de la sesión. Es un método más fiable que usar *cookies*, ya que algunos navegadores web están configurados para rechazarlas. Sin embargo, desde el punto de vista del programador es un método de mayor complejidad y menor elegancia.

Como inconveniente, si el servidor al que un usuario es redirigido se cae, la información de sesión se pierde. Se produce el mismo problema si se cuenta con dos o más balanceadores de respaldo y el principal se cae, la tabla que relaciona la sesión entre usuarios y servidores se perderá.