# Alta dispoñibilidade en servidores Web

Sempre que teñamos un servidor web replicado en varios equipos, podemos engadir varios rexistros tipo A apuntando aos diferentes enderezos IP, para que poidan respostar as peticións a unha URL referida a un host virtual.

Exemplo:

> Podemos ter o host virtual "exemplo.com" replicado nos equipos con enderezo 192.168.0.100 e 192.168.0.101. Se configuramos no servidor DNS dous rexistros tipo A
>
> ```
> sitio.lan A 192.168.0.100
> sitio.lan A 192.168.0.101
> ```
>
> conseguiremos que as peticións HTTP sexan atendidas por cadanseu equipo de forma alternada. Se queremos, que as peticións estén balanceadas nun 50%, podemos engadir a maiores rexistros SRV
>
> ```
> sitio.lan A 192.168.0.100
> sitio.lan A 192.168.0.101
> _http._tcp SRV 10 60 80 sitio.lan
> _http._tcp SRV 10 40 80 sitio.lan
> ```

Pero, nunca conseguiremos que se un equipo está caido, os demáis asuman as peticións HTTP, senón que os clientes seguirán esperando a que este responda.

Para solucionar este tipo de problemas, idearonse os **Proxy**.

## Sumario

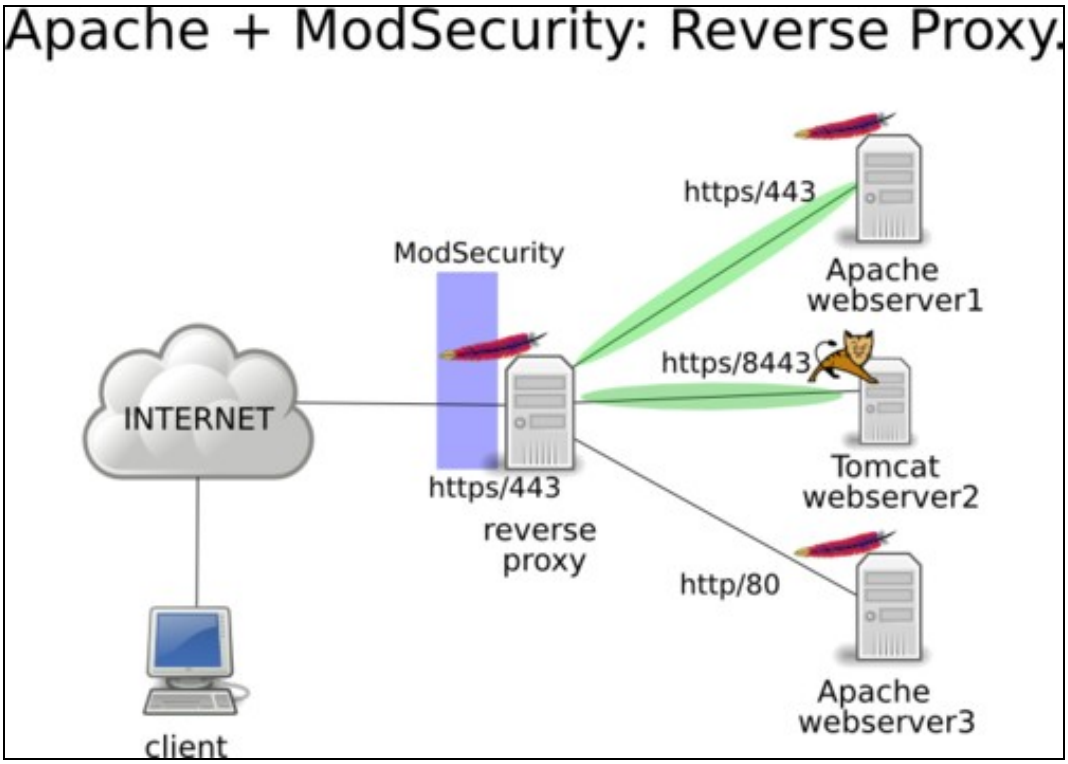


## Proxy

Un *Proxy* é unha porta de acceso para os usuarios, que actúa de intermediario entre estes e os servidores web que atenden as peticións HTTP. Os usuarios configuran o seu proxy nos seus axustes do navegador, e todas as peticións de HTTP son remitidas a ese proxy que se encargará de solicitar a petición HTTP ao servidor web correspondente, e retransmitirlla ao usuario. Podemos usar proxies para:

- acelerar o acceso a Web, facendo caché de páxinas, así as páxinas moi concorridas non teñen que ser descargadas de todas as veces.
- habilitar acceso controlado a web detrás dun firewall
- filtrar ou transformar condito web

Un **Proxy inverso** é unha porta de acceso para os servidores, e permite que un servidor web provea contido para outro de forma transparente. Mediante proxies inversos, podemos aumentar o rendemento web, facendo caché de peticións HTTP, e tamén para permitir replicara servidores web facendo balanceo de carga, ou protexendo a outros que poden ser vulnerables. Pero o uso máis habitual é habilitar acceso controloado desde a Web a servidores que están detrás dun firewall.




## Proxies (inversos) con Apache

Os módulos de Apache que permiten o uso de proxies inversos, son:

- **mod_proxy** para todo tipo de proxies (módulo base).
- **mod_proxy_http** para proxies inversos co protocolo HTTP (require que *mod_proxy* esté activo.

Unha vez activados os dous módulos, xa podemos empregar as directivas **ProxyPass** e **ProxyPassReverse**.

Vémolo cun exemplo:

```
<VirtualHost *:80>
ServerName www.example.com
ProxyRequests Off
ProxyPreserveHost On
ProxyPass / http://foo.example.com/  #Ou enderezo IP
ProxyPassReverse / http://foo.example.com/ #Ou enderezo IP
</VirtualHost>
```

Neste caso, sempre que accedamos a *http://www.example.com/* accederemos realmente a *http://foo.example.com/* e no exemplo

```
<VirtualHost *:80>
ServerName www.example.com
ProxyRequests Off
ProxyPreserveHost On
ProxyPass /foo http://foo.example.com/bar
ProxyPassReverse /foo http://foo.example.com/bar
</VirtualHost>
```

sempre que accedamos a *http://www.example.com/foo* accederemos realmente a *http://foo.example.com/bar*

A directiva *ProxyRequests Off* evita que o front-end sea empregado como proxy, é dicir, que os usuarios poidan saltar ao front-end y de ahí a cualquera outro enderezo. É moi importante deixalo deshabilitado para evitar problemas de seguridade ou incluso legais.

A directiva *ProxyPreserveHost On* permite que o salto do servidor de front-end ao de back-end sea transparente para o usuario. Se non estivera habilitada, o usuario dirixiríase a http://www.example.com pero inmediatamente vería como o enderezo cambia a http://foo.example.com. Ademáis, como neste suposto o servidor de back-end non é visible desde Internet o usuario vería un erro.

Por último, as directivas *ProxyPass* e *ProxyPassReverse* xestionan o salto e a volta do servidor de front-end ao de back-end.

### Proxy apache balanceado

Se ademáis queremos que o proxy faga balanceo de carga entre varios servidores, entón teremos que habilitar o módulo **mod_proxy_balancer** e tamén algún balanceador como pode ser **mod_lbmethod_byrequests**

O seguinte exemplo, amosa un proxy balanceado

```
ServerName www.example.com
ProxyRequests Off
ProxyPreserveHost On
<Proxy balancer://mycluster>
    BalancerMember http://eq1.example.com
    #Ou enderezo IP
    BalancerMember http://eq2.example.com
    #Ou enderezo IP
</Proxy>
ProxyPass / balancer://mycluster/
ProxyPassReverse / balancer://mycluster/
```

Neste caso, sempre que se acceda a example.com, as peticións serán repartidas entre eq1.example.com e eq2.example.com

Existen multitude máis de configuracións, para traballar con sesións e máis características. Recoméndase consultar a documentación oficial.

## Varnish

Varnish é un proxy HTTP inverso, tamén coñecido como acelerador web ou acelerador HTTP. Almacena ficheiros e fragmentos de arquivos na memoria, permitindo que poidan ser servidos moi rápido.

Pero a súa vez tamén é flexible, e sinxelo de configurar. É código aberto e software libre, con desenrolo público e licencia BSD.

O único punto negativo é que non soporta HTTP

### Instalación

Varnish ven de serie nos repositorios de Ubuntu/Debian, pero a versión que se instala está un pouco desactualizada. Imos instalalo a partires dos repositorios. **Debian**

```
apt-get install debian-archive-keyring curl gnupg apt-transport-https
curl -L https://packagecloud.io/varnishcache/varnish60lts/gpgkey | apt-key add -
```

#### Debian

```
echo 'deb https://packagecloud.io/varnishcache/varnish60lts/debian/ buster main' | tee /etc/apt/sources.list.d/varnish.list >/dev/nul
apt-get update
apt-get install varnish
```

#### Ubuntu 20.04

```
echo 'deb https://packagecloud.io/varnishcache/varnish60lts/ubuntu/ focal main' | tee /etc/apt/sources.list.d/varnish.list >/dev/null
apt-get update
apt-get install varnish
```

### Configuración

Varnish pode ser configurado para traballar en modo comando, ou mediante ficheiros de configuración (Ficheiros VCL). Sempre que se faga un cambio, é necesario reiniciar o servidor.

**Equipos que funcionan con *init.d***

```
service varnish restart
```

**Equipos que funcionan con *systemd***

```
systemctl restart varnish.service
```

Os ficheiros de configuración principais son:

- */etc/varnish/default.vcl*: Ficheiro VCL principal. Contén toda a configuración do servidor.

Na configuración do ficheiro VCL, temos xa pre-configurado un servidor proxy, cara o propio equipo local, no porto 8080. Na terminoloxía varnish, un **backend** é un equipo cun servidor web do que este servidor varnish é proxy inverso.

```
backend default {
    .host = "127.0.0.1";
    .port = "8080";
}
```

Con esto, teremos definido o enderezo IP e o porto do equipo remoto. Falta o porto de escoita de varnish, que definimos de diferentes maneiras según o equipo funcione con *init.d* ou *systemd* en */etc/default/varnish* ou */lib/systemd/system/varnish.service*

Para *init.d* **Debian/Ubuntu (legacy)**

```
#/etc/default/varnish
DAEMON_OPTS="-a :80 \
             -T localhost:6082 \
             -f /etc/varnish/default.vcl \
             -S /etc/varnish/secret \
             -s malloc,256m"
```

Para *systemd* **Debian (v8+)** / **Ubuntu (v15.04+)**

```
#/lib/systemd/system/varnish.service
[Unit]
Description=Varnish Cache, a high-performance HTTP accelerator

[Service]
Type=forking

# Maximum number of open files (for ulimit -n)
LimitNOFILE=131072

# Locked shared memory (for ulimit -l)
# Default log size is 82MB + header
LimitMEMLOCK=82000

# On systemd >= 228 enable this to avoid "fork failed" on reload.
#TasksMax=infinity

# Maximum size of the corefile.
LimitCORE=infinity

# Set WARMUP_TIME to force a delay in reload-vcl between vcl.load and vcl.use
# This is useful when backend probe definitions need some time before declaring
# configured backends healthy, to avoid routing traffic to a non-healthy backend.
#WARMUP_TIME=0

ExecStart=/usr/sbin/varnishd -a :80 -T localhost:6082 -f /etc/varnish/default.vcl -S /etc/varnish/secret -s malloc,256m
ExecReload=/usr/share/varnish/reload-vcl

[Install]
WantedBy=multi-user.target
```

No caso de que o equipo traballe con systemd, hai que activar o servizo para que arranque ao iniciar o equipo.

```
systemctl daemon-reload
```

```
systemctl restart varnish.service
```

### Multiple Host virtuais

Tamén podemos definir varios backend

```
#/etc/varnish/default.vcl
backend foo {
    .host = "192.168.0.100";
    .port = "8080";
}

backend bar {
    .host = "192.168.0.101";
    .port = "8080";
}
sub vcl_recv {
    if (req.http.host ~ "foo.com") {
        set req.backend_hint = foo;
    } elsif (req.http.host ~ "bar.com") {
        set req.backend_hint = bar;
    }
}
```

### Balanceo de carga

Tamén podemos facer balanceo de carga

```
#/etc/varnish/default.vcl


backend server1 {
    .host = "server1.example.com";
    .probe = {
        .url = "/test.html";
        .timeout = 1s;
        .interval = 5s;
        .window = 5;
        .threshold = 3;
    }
}

backend server2 {
    .host = "server2.example.com";
    .probe = {
        .url = "/test.html";
        .timeout = 1s;
        .interval = 5s;
        .window = 5;
        .threshold = 3;
    }
}
import directors;
sub vcl_init {
    new vdir = directors.round_robin();
    vdir.add_backend(server1);
    vdir.add_backend(server2);
}
```

Se so temos un backend default, asignamoslle todas as peticións

```
sub vcl_recv {
    # send all traffic to the bar director:
    set req.backend_hint = vdir.backend();
}
```

E se temos máis dun "Virtual Host", asignamoslle o backend correcto dependendo do nome do servidor.

```
sub vcl_recv {
    if (req.http.host ~ "foo.com") {
```

```
        set req.backend_hint = foo;
    } elsif (req.http.host ~ "bar.com") {
        set req.backend_hint = vdir.backend();
    }
}
```

Neste exemplo, conseguimos que a carga se reparta entre os dous servidores seguindo o algoritmo round-robin, anque debido a que garda os datos na caché, é probable que non poidamos testear como reparte as conexións. Simplemente veremos que atende as peticións pero que por un longo periodo de tempo, non lle fai peticións aos servidores "backend".

Se queremos eliminar a opción de manexo de sesións "*pegañentas*" podemos facelo seguindo esta guía:

Podemos ver o estado de "saúde" dos servidores backend co comando

```
varnishlog -g raw
```

## Nginx

Nginx é un servidor HTTP caché lixeiro que tamén se pode empregar como proxy inverso. O máis usual é configuralo como proxy inverso de HTTPS.

Nginx é o mellor servidor web para servir arquivos estáticos, e ademais faio tendo un consumo de memoria considerablemente menor que Apache, dado que non precisa crear un proceso ou fío novo para cada pedido que lle chega, e tampouco necesitar analizar todos os directorios dun sitio buscando arquivos **.htaccess**, xa que carga as regras de Rewrite directamente desde a súa configuración.

O primeiro paso é instalar o nginx

```
apt-get install nginx
```

O seguinte paso será cambiar os portos de escoita de Apache

E por último, cremos un novo virtual host para nginx. Por exemplo **/etc/nginx/sites-available/exemplo.com**

```
#/etc/nginx/sites-available/exemplo.com
server {
    listen 80;
    server_name  www.exemplo.com;
    location / {
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://eq1.exemplo.com:8080;  #Tamen podemos poñer o enderezo IP
    }
}
```

Tamén o podemos facer con HTTPS:

```
#/etc/nginx/sites-available/exemplo.com

server {

    listen 443 default_server;
    server_name www.example.com;

    ssl on;
    ssl_certificate /usr/local/nginx/conf/server.pem;
    ssl_certificate_key /usr/local/nginx/conf/server.key;
    ssl_session_cache shared:SSL:10m;

    location / {
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://eq1.exemplo.com:8080;
    }
```

```
}
```

### Balanceador de carga

Se queremos facer un balanceador de carga con *nginx* podemos empregar unha configuración similar á seguinte:

```
upstream myapp1 {
    server srv1.example.com; #Tamen podemos poñer o enderezo IP directamente.
    server srv2.example.com;
    server srv3.example.com;
}

server {
    listen 80;

    location / {
        proxy_pass http://myapp1;
    }
}
```

Se non se expresa ningún método de balanceo, enténdese que por defecto se emprega *round-robin*

Tamen podemos repartir a carga con pesos específicos:

```
upstream myapp1 {
    server srv1.example.com weight=3;
    server srv2.example.com;
    server srv3.example.com;
}
```

asignala ao servidor con menos conexións

```
upstream myapp1 {
    least_conn;
    server srv1.example.com;
    server srv2.example.com;
    server srv3.example.com;
}
```

E tamén podemos facer que as sesións sexan persistentes. Un mesmo cliente, namentres a sesión permaneza activa, sempre será atendido por un mesmo servidor backend. Enténdese que a sesión é deste xeito pegañenta.

```
upstream myapp1 {
    ip_hash;
    server srv1.example.com;
    server srv2.example.com;
    server srv3.example.com;
}
```

## Balanceador de carga de nivel de aplicación HAProxy

HAProxy é un balanceador de carga específico de HTTP e HTTPS de nivel de aplicación (L7), anque tamén pode traballar a nivel de rede (L4). Traballar a nivel de aplicación abre novas posibilidades ó poder acceder á capa máis próxima ó usuario. Centrándonos na web, poderemos tomar decisións en funcións das cabeceiras http dos paquetes, insertar cookies, marcar paquetes, ...; e polo tanto, poderemos reenviar as solicitudes dos clientes a diferentes servidores en base ó contido da petición do cliente. Un dos usos máis interesantes é a posibilidade de correr diferentes aplicacións en diferentes servidores reais pero accesibles baixo o mesmo nome de dominio e porto. Entre outras cousas,tamén permite establecer os certificados para HTTP no frontend, permitindo que as comunicacións entre cliente e proxy, vaian encrintadas con HTTPS e entre o frontend e backend mediante HTTP.

## Instalación HA Proxy

O primeiro paso para instalr o HAProxy é engadir os repositorios: **Debian 10**

```
apt update
apt install curl gnupg debian-archive-keyring apt-transport-https

curl https://haproxy.debian.net/bernat.debian.org.gpg | apt-key add -
echo deb http://haproxy.debian.net buster-backports-2.3 main | \
      tee /etc/apt/sources.list.d/haproxy.list
apt update
apt install haproxy=2.3.\*
```

**Ubuntu 18.04**

```
apt-get install software-properties-common
add-apt-repository ppa:vbernat/haproxy-2.1
apt-get update
apt-get install haproxy
```

## Configuración

Teremos os ficheiros de configuración en */etc/haproxy/*

Para configurar HAproxy hai tres elementos fundamentais:

- ACLs (Acces Control List): permiten comprobar algunha condición e en base ó resultado realizarase unha acción.
- Seccións Backend: definen un conxunto de servidores que recibirán as solicitudes reenviadas por HAProxy.
- Seccións Frontend: definen como se reenvían as solicitudes ós backends.

```
frontend sitio_http
   bind 192.168.0.20:80
   acl host_www.proba.lan hdr(host) -i proba.lan
   acl host_www.proba.lan hdr(host) -i www.proba.lan
   use_backend proba_lan_back if host_www.proba.lan


backend proba_lan_back
   balance roundrobin
   server server01 192.168.1.41:80 check
   server server02 192.168.1.42:80 check
```

No caso de querer establecer pesos entre os distintos backend, introducímolos da seguinte maneira, tendo en conta que o peso é un número entre 0 e 256

```
frontend sitio_http
   bind 192.168.0.20:80
   acl host_www.proba.lan hdr(host) -i proba.lan
   acl host_www.proba.lan hdr(host) -i www.proba.lan
   use_backend proba_lan_back if host_www.proba.lan


backend proba_lan_back
   balance roundrobin
   server server01 192.168.1.41:80 check weight 22
   server server02 192.168.1.42:80 check weight 40
```

Unha vez feito isto e reinciado o servizo haproxy, aparecerán no ficheiro de log (*/var/log/haproxy.log*) todas as peticións que se fagan ao balanceador HAProxy.

### HAProxy e HTTPS

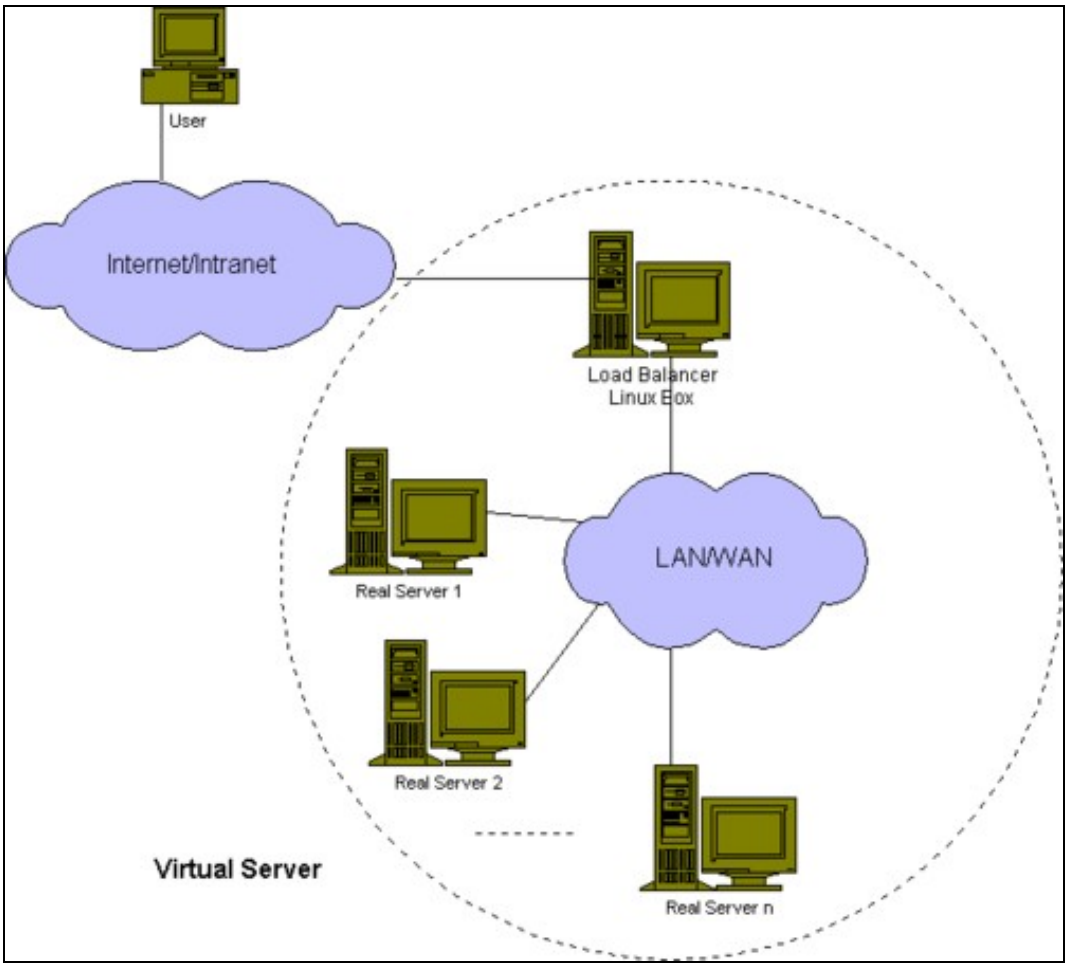
Para configurar HAProxy con HTTPS necesitaremos que o certificado do sitio e a chave privada estén no mesmo ficheiro con extensión **.pem**.

So queda engadir unha sección frontend para empregar HTTPS:

```
frontend sitio_https
   bind 192.168.57.40:443 ssl crt /etc/ssl/private/proba.lan.pem
   acl host_www.proba.lan hdr(host) -i proba.lan
   acl host_www.proba.lan hdr(host) -i www.proba.lan
   use_backend proba_lan_back if host_www.proba.lan
```

## Proxy e Balanceador a nivel de rede con Linux Virtual Server (LVS)

O Linux Virtual Server é un servidor altamente scalable e altamente dispoñíbel construído formando un grupo de servidores reais ou cluster. A arquitectura do cluster de servidores é plenamente transparente para os usuarios finais. Estes, interactúan co sistema de cluster como se era só fose un so. Sirva coma exemplo a seguinte figura.



Os servidores reais e o balanceador de carga poden estar intercontectados mediante unha LAN de alta velocidade ou dispersos nunha WAN. Os balanceadores poden atender peticións para diferentes servidores e aparentar que funciona coma un servizo único nun único enderezo IP. Para atender as peticións pode empregarse un balanceo a nivel de rede ou a nivel de aplicación. A escalabilidade do sistema consíguese de xeito transparente engadindo ou quitando nodos no cluster. A alta dispoñibilidade conseguese detectando a caída de nodos e reconfigurando o sistea de xeito apropiado.

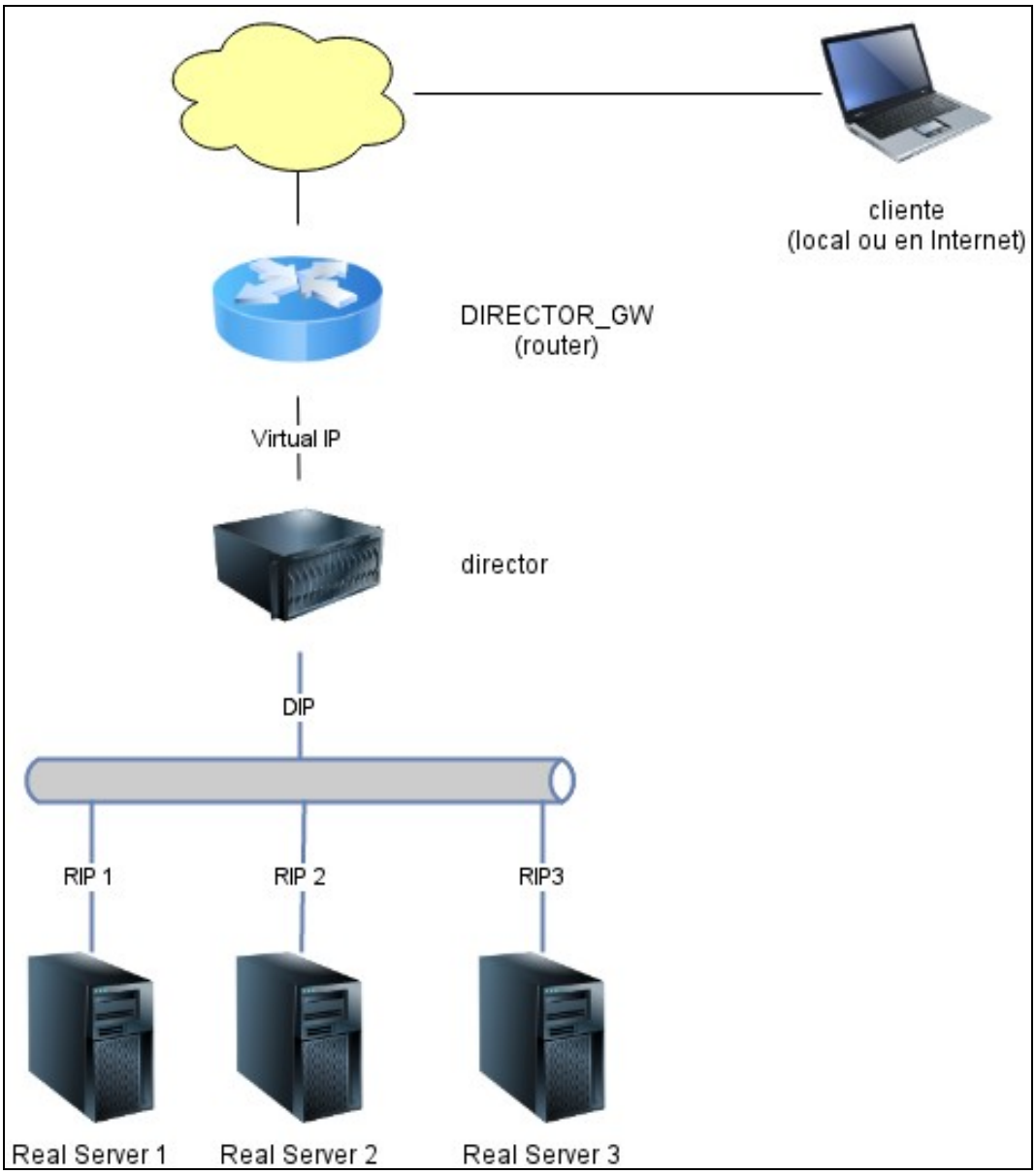
O Linux Virtual Server Project (LVS) implementa balanceo a nivel 4 (tcp/udp) no kernel de Linux. LVS corre sobre Linux pero é capaz de balancear conexións procedentes de usuarios finais sobre calquera sistema operativo a servidores reais correndo en calquera sistema operativo. Mentres as conexións sexan tcp ou udp, LVS pode usarse. Esto significa nin máis nin menos, que se temos un sistema Linux temos a nosa disposición un balanceador.

Neste sistema destacamos dous compoñentes pola súa función:

- **IPVS** (IP Virtual Server): é o compoñente integrado no kernel de Linux que se encarga do balanceo de carga.
- **ldirectord**: é un servizo encargado de monitorar os servidores reais dun balanceador e no caso de fallo dalgún deles, encárgase de reconfigurar dinámicamente IPVS.

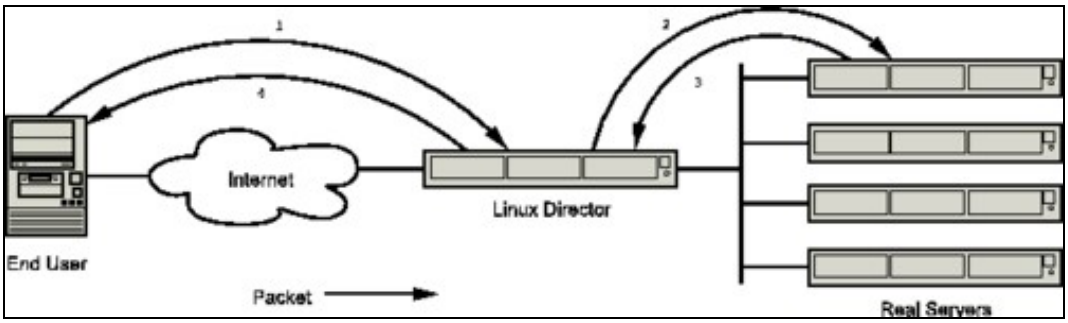
Nun sistema baseado en LVS adoitase a usar a seguinte terminoloxía:

- **Linux Director**: equipo correndo Linux con LVS instalado que recibe paquetes dende usuarios finais e os reenvía cara ós servidores reais.
- **End User**: equipo que orixina a conexión.
- **Real Server**: equipo final que corre algún servizo tipo servidor web.
- **Cliente IP Address (CIP)**: o enderezo IP do cliente que orixina a conexión.
- **Virtual IP Address (VIP)**: o enderezo IP asignado a un servizo que xestiona o director.
- **Real IP Address (RIP)**: o enderezo IP dun Real Server.
- **Director IP Address (DIP)**: o enderezo IP do director na rede DRIP.

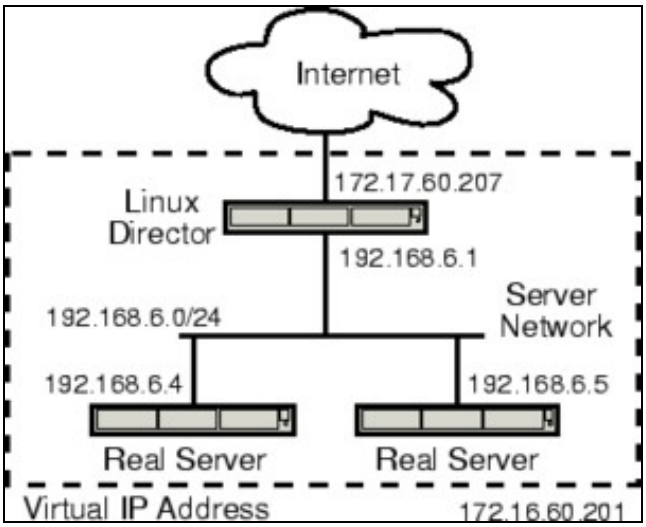

cliente
(local ou en Internet)
DIRECTOR_GW
(router)
Virtual IP
director
DIP
RIP 1
RIP 2
RIP3
Real Server 1
Real Server 2
Real Server 3

LVS ten varias formas diferentes de reenviar paquetes cara ós servidores reais sendo as principais:

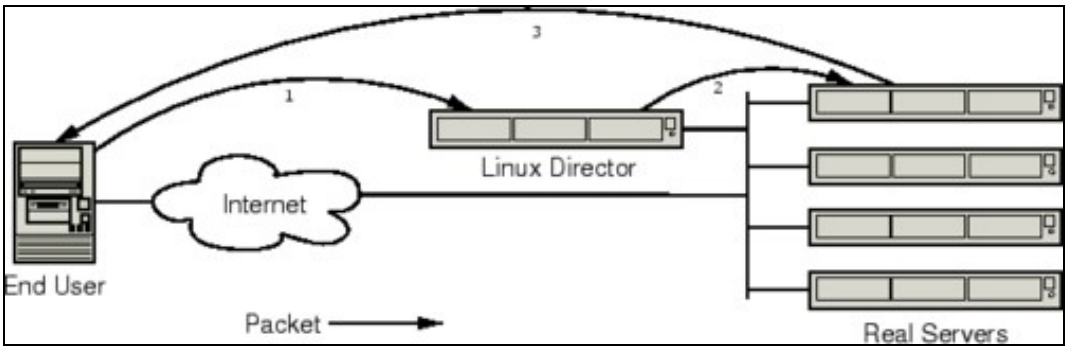
- **LVS Network Address Translations (LVS-NAT)**: Os paquetes recíbense dos usuarios finais e os enderezo IP/porto destino cámbianse polos do servidor real seleccionado. Os paquetes de resposta pasan a través do director e volven modificarse para que o usuario final reciba unha resposta desde a orixe esperada. Debido ó traballo a desempeñar polo balanceadore e á súa ubicación; este pode converterse nun problema, debido a carga de CPU que supón realizar o NAT e o ancho de banda real das nosas interfaces de rede; xa que, polo balanceador van pasar tanto as peticións cara ós servidores como as respostas cara ós clientes.



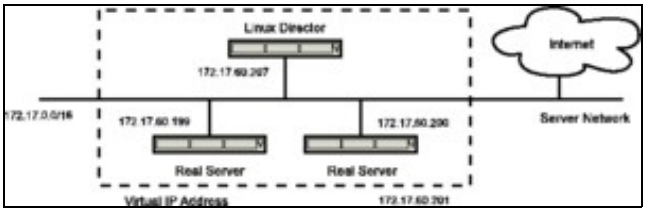

Nun caso práctico teriamos o seguinte:



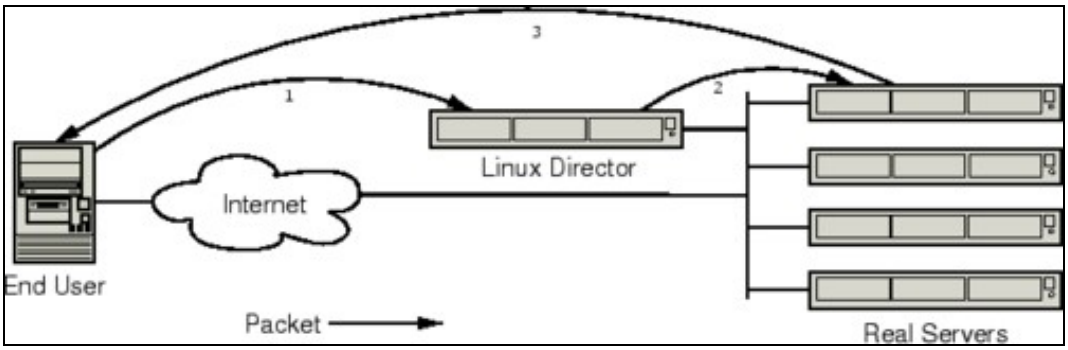

- **LVS Direct Routing (LVS-DR)**: Os paquetes dos usuarios finais reenvíanse directamente cara ó servidor real seleccionado sen modificar a IP (polo tanto os servidores reais deben configurarse para aceptar paquetes con destino á VIP). Os servidores reais responden directamente ós usuarios finais, polo que o director queda liberado desa función aforrándose CPU e ancho de banda.



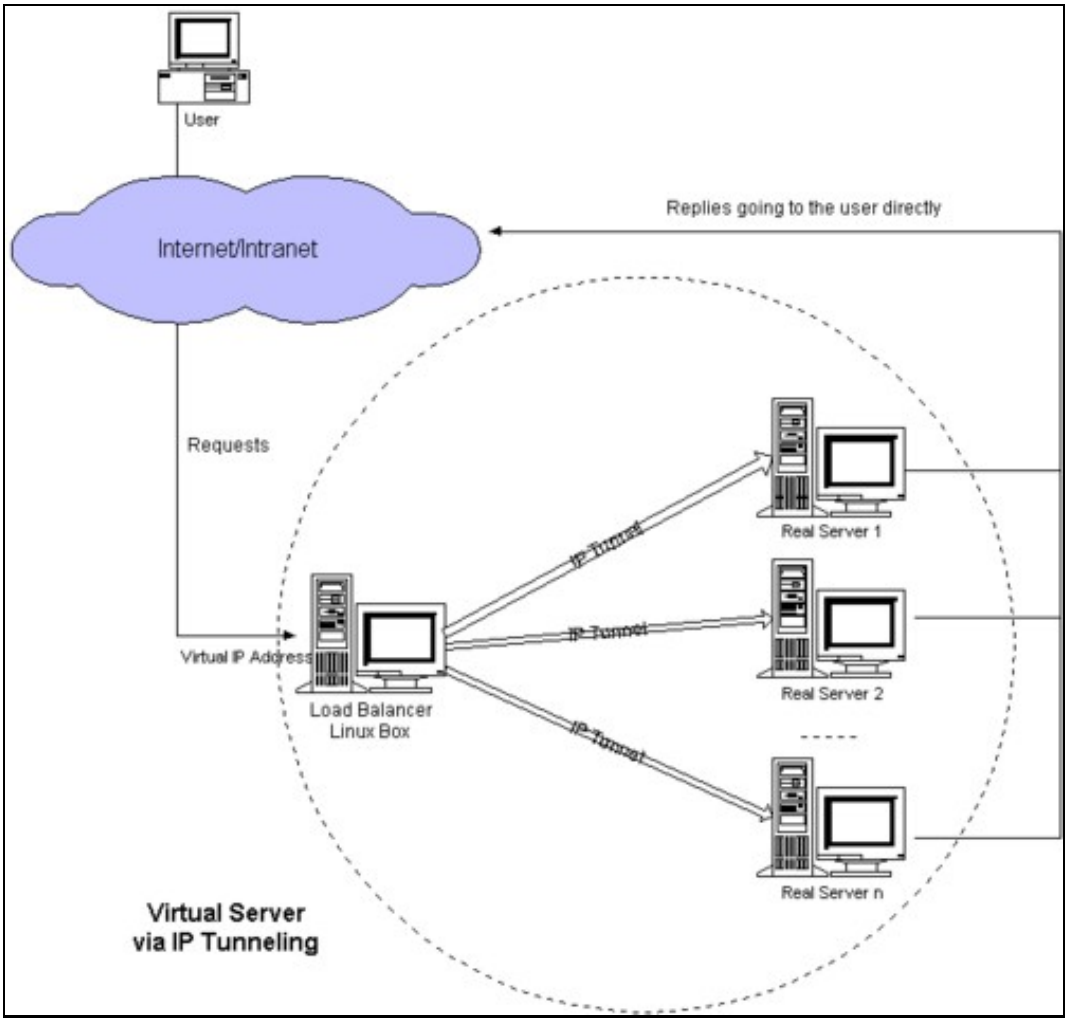
Nun caso práctico teriamos o seguinte:




- **LVS IP-IP Encapsulation (LVS-Tunneling)**: é semellante ó LVS-DR, pero os servidores reais poden estar en redes diferentes (mesmo en localizacións xeográficas diferentes), polo que os paquetes encapsúlanse dentro de paquetes IP no canto de tramas Ethernet.

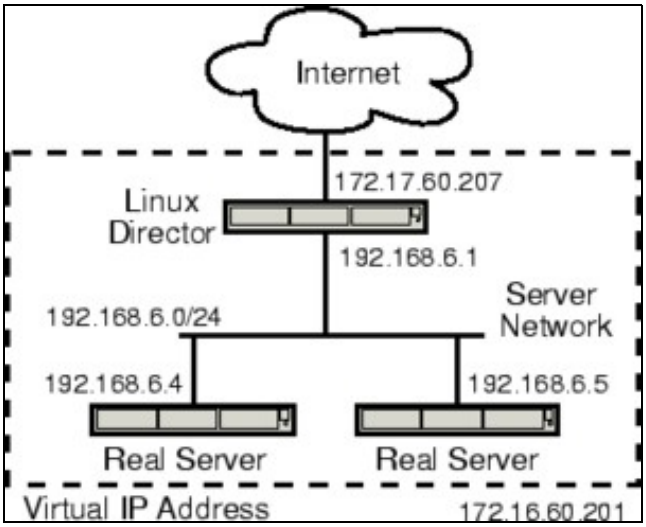



Nun caso práctico teriamos o seguinte:



### Implementación de LVS e NAT

Partiremos do modo de funcionamento de LVS - NAT, na que o equipo frontend, tamén fará de router dos servidores reais ou equipos backend.




Para instalar ldirectord

```
apt-get install ldirectord
```

Cando un usuario accede ó servizo proporcionado polo cluster, o paquete da solicitude vai destinado enderezo IP Virtual (VIP) e chega ao interface WAN do balanceador. O balanceador examina o enderezo IP/porto destino; e no caso de coincidir cun servizo configurado, procede a escoller un dos servidores reais e reescribir o enderezo IP/porto destino polos correspondentes valores asociados ó servidor real escollido. Esta conexión métese nunha táboa de conexións para facer o seu seguimento. Cando paquete de resposta procedente do servidor real chega ó balanceador, reescribirase o enderezo IP/porto orixe reemprazando os datos do servidor real polos do VIP.

O equipo LVS-NAT ademais de xestionar a reescritura dos paquetes ten que enrutalos; polo tanto, hai que activar o enrutamento IP asignando á variable do sistema *net.ipv4.ip_forward* o valor 1. Para asignarlle o valor 1 e facer o cambio persistente, resistinto polo tanto reinicios da máquina, editaremos o arquivo de configuración /etc/sysctl.conf e descomentados a liña correspodente:

```
# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
```

E para aplicar os cambios sen reiniciar

```
sysctl -p
```

A continuación configuraremos *ldirectord*, editando o arquivo **/etc/ldirectord.cf**. As opcións de configuración poden consultarse nas páxinas de manual de *ldirectord* e *ipvsadm*:

**ldirectord como Proxy**

Configuración común:

```
checktimeout=3
checkinterval=5
failurecount=3
autoreload=no
logfile="/var/log/ldirectord.log"
quiescent=no
cleanstop=yes
```

Configuración do proxy

```
virtual=172.17.60.207:80
    real=192.168.6.4:80 masq
    protocol=tcp
    scheduler=rr
    # monitorización do servidor real
    service=http
```

```
    request="test.html"
    receive="OK"
```

Como podemos comprobar, para saber se o servidor real está funcionando, realiza a descarga periódica do arquivo "test.html" esperando que como resposta apareza "OK" cada 5 segundos. Ao terceiro fallo, márcao como "non activo"

Reiniciamos o servizo *ldirectord*:

```
service ldirectord restart
```

Podemos comprobar no ficheiro de log indicado, que se consegui conectar co equipo remoto.

**ldirectord como Balanceador**

Podemos configurar o balanceador co número de servidores que queiramos empregando o algoritmo "round robin" para repartir a carga, e adicionalmente cun servidor "fallback" ao que reenviar os paquetes no caso de que non esté activo ningún dos servidores reais.

```
virtual=172.17.60.207:80
    real=192.168.6.4:80 masq
    real=192.168.6.5:80 masq
    fallback=192.168.6.100:80 masq
    protocol=tcp
    scheduler=rr
    # monitorización do servidor real
    service=http
    request="test.html"
    receive="OK"
```

Tamén podemos asignar pesos á hora de repartir a carga:

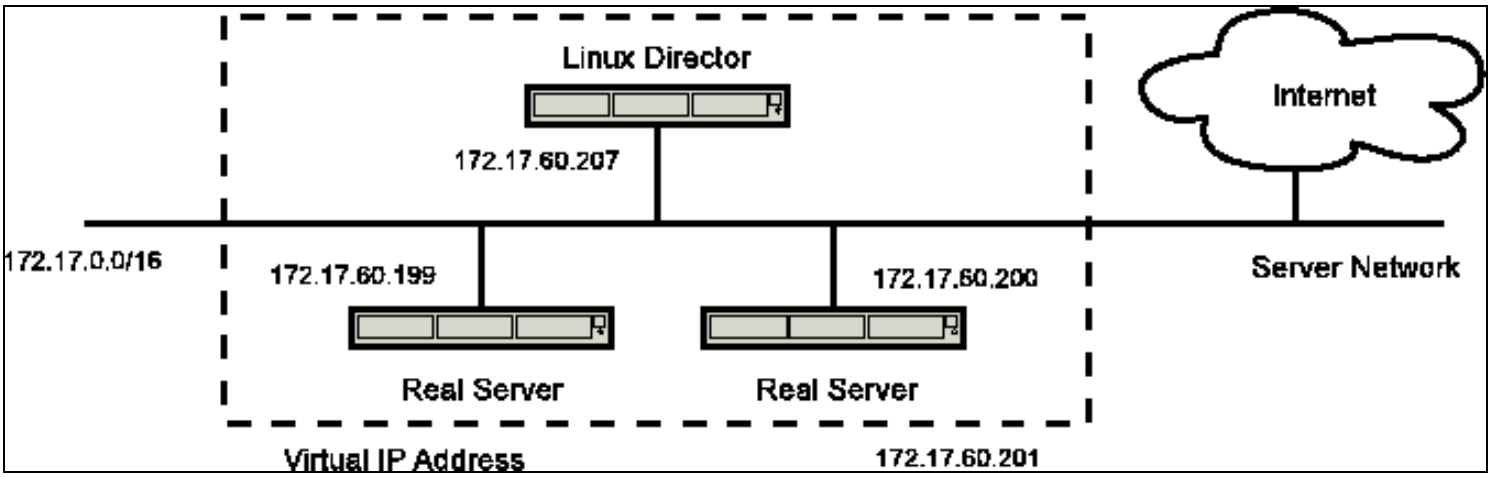
```
virtual=172.17.60.207:80
    real=192.168.6.4:80 masq 4
    real=192.168.6.5:80 masq 2
    real=192.168.6.5:80 masq 1
    fallback=192.168.6.100:80 masq
    protocol=tcp
    scheduler=wrr
    # monitorización do servidor real
    service=http
    request="test.html"
    receive="OK"
```

Despois de facer unhas cantas conexións desde calquera cliente, podemos ver as estatísticas de conexións executando o comando **ipvsadm -L -n**:

```
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port          Forward Weight ActiveConn InActConn
TCP  172.17.60.207:80 rr
  -> 192.168.6.4:80              Masq    1      0          2
  -> 192.168.6.5:80              Masq    1      0          2
  -> 192.168.6.6:80              Masq    1      0          2
```

## Implementaión de LVS e DR

Partiremos do modo de funcionamento de LVS - DR, na que o equipo frontend, xa non fará de router dos servidores reais ou equipos backend. Neste caso, todos os equipos terán un enderezo virtual común para todos eles.




Para poñer ese enderezo común a todos eles (frontend e backends), podemos facelo do seguinte xeito no ficheiro /etc/network/interfaces

```
auto eth0:0
iface eth0:0 inet static
    address 172.17.60.201/16
```

A continuación configuramos o ldirectord

```
checktimeout=3
checkinterval=5
failurecount=3
autoreload=no
logfile="/var/log/ldirectord.log"
quiescent=no
cleanstop=yes

virtual=172.17.60.201:80
        # servidores reais balanceados en modo round-robin
        real=172.17.60.200:80 gate
        real=172.17.60.199:80 gate
        #    fallback=172.17.60.210:80 gate
        protocol=tcp
        scheduler=rr
        checktype=negotiate
        # monitorización dos servidores reais
        service=http
        request="test.html"
        receive="OK"
```