



100% Libre de M\$

Historia | TuxTips | Artículos | Eventos | Screenshots | Info | Links | Contacto

Principal > Artículos >

Configuración de un WebCluster

Juan Carlos Inostroza
jci@tux.cl

A diferencia de otros tipos de clusters, como MOSIX, un Webcluster es una colección de máquinas que sirven páginas WEB. Estas máquinas se unen dentro de una pequeña "granja" o familia de servidores. Gracias a algunas escondidas pero poderosas potencialidades de Apache se puede lograr esto último.

1. Explicación Técnica

Apache posee un módulo, **libproxy.so**, que posee la misma funcionabilidad que un proxy. Un Proxy es una máquina instalada en una red cuya función es conectarse a otra red, obtener información y entregarla a los clientes conectados al Proxy. Un proxy no es un Gateway ni un Router, aunque generalmente se le confunde. Un Router o un Gateway desvían paquetes TCP/IP de acuerdo a una serie de reglas, llamadas generalmente reglas de enrutamiento. Estas reglas funcionan en el nivel más básico de TCP/IP. Un proxy es un como un "router de alto nivel". No desvía paquetes y funciona como un intermediario de contenidos entre un cliente y una red.

```
[cliente1]----+
[cliente2]----+----[ PROXY ]----[INTERNET]
[cliente3]----+

      Cliente-(+)->PROXY
                PROXY --GET(*)---->WWW SERVER
                PROXY <-----(*)---WWW SERVER
      Cliente<--(*)PROXY
```

Un servidor Web de una intranet, la gran mayoría de las veces esta conectado a Internet. Por ejemplo, la empresa X posee una intranet y un servidor Web que sirve al dominio www.x.cl. Si un cliente externo desea acceder a la intranet de la empresa X (digamos, legalmente =) para revisar estadísticas internas, contactarse con algún agente, etc, debe haber un Router que haga que máquinas de la intranet "salga" de la intranet y sea visible desde el resto de Internet. Uno de los paquetes usados regularmente para esto es IPTABLES usando NAT. Esto no es difícil, pero siempre existe el problema que debe existir un Router por obligación.

Volviendo a libproxy.so, este modulo permite que Apache haga una conexión entre dos máquinas, internet e intranet, de manera totalmente transparente para el cliente, haciendo más fácil el trabajo y así eliminando la necesidad de un Router.

```
                +---[ INTRANET SERVER1 ]
[INTERNET]-----[ WWW ]-----+---[ INTRANET SERVER2 ]
                +---[ INTRANET SERVER3 ]

[INTERNET] GET----->WWW
                WWW (URL_RW) (*)
                WWW --- (GET) (P) (*)-----[ INTRANET SERVERX ]
                WWW <----- (*)
[INTERNET] <---- (*) ---WWW
```

Como se ve en la figura, el servidor WWW hace una conexión vía HTTP a un computador de la intranet. Esto hace que las paginas FÍSICAMENTE no residan en el servidor, sino en un computador de la Intranet. (que es justamente la tarea de un proxy). Este tipo de conexión es totalmente transparente para el cliente.

Aquí es donde entra otro modulo de Apache. Se llama **mod_rewrite**. Su tarea principal es reescribir las peticiones de acuerdo a cierto parámetro de búsqueda.

1.1 Reescribiendo URL (URL Rewrite)

Hay dos maneras de reescribir un URL:

- Client Rewriting: esta es la forma menos transparente de todas. Si un cliente HTTP se conecta, por ejemplo, a www.foo-bar.com, el servidor puede responder de varias maneras para reescribir el URL:
 - usando Javascript y document.location
 - usando CGIs
 - HTML y contenidos META (Refresh)
 - reescribiendo las cabeceras usando PHP y la función header()

Suponiendo el mismo ejemplo:

- [1] Cliente se conecta a http://www.foo-bar.com
- [2] El servidor responde con un index.php
- [3] El contenido de index.php es <? header ("Location: http://freehosting.foo.bar/users/free/dir1/index.html"); ?>
- [4] El cliente es redirigido a http://freehosting.foo.bar/users/free/dir1/index.html

- Server Rewriting: Es la forma más transparente. En vez de reescribir el URL en el cliente, el URL es transformado DENTRO del servidor. El servidor reinterpreta la conexión, hace la conexión donde FÍSICAMENTE residen las páginas y entrega el resultado al cliente.

Suponiendo el mismo ejemplo de www.foo-bar.com:

- [1] El cliente se conecta a http://www.foo-bar.com
- [2] El servidor detecta que debe haber una reescritura de URL
- [3] Como el cliente no pide ninguna pagina, asume que es la pagina indice configurada en Apache o en su VirtualHost En este caso, index.html
- [4] El servidor reinterpreta el URL : http://www.foo-bar.com/* = http://freehosting.foo.bar/users/free/dir1/*
- [5] El servidor se conecta a http://freehosting.foo.bar/users/free/dir1/index.html
- [6] Reacomoda las cabeceras y los contenidos del HTML para que pareciera que la conexión no sale del servidor
- [7] Entrega la pagina index.html

El cliente no sabe que las paginas estan en freehosting.foo.bar/users/free/dir1 y todas las conexiones del cliente son hacia http://www.foo-bar.com.

1.2 Está bien, pero de qué sirve?

(...la pregunta de mi jefe)

1.2 Esta bien, pero de que sirve?

(...la pregunta de mi jefe)

Esto brinda las siguientes soluciones:

- Olvidarse de aprender IPTABLES y NAT para que una máquina de la intranet se vea desde afuera
- Poder mantener servidores dentro de una intranet o una DMZ
- Poder realizar una granja de servidores donde todas las conexiones se ven de manera transparente desde una intranet, DMZ o internet

La última solución propone el asunto de tener un Webcluster. Es decir, muchas máquinas sirviendo una sola página. Y la tarea de reescribir los URL se le encarga al servidor central.

```
[Nodo1]----+
[Nodo2]----+
[Nodo3]----+----<Cluster0 (Central)>----[INTERNET]
  ...
[Nodo_n]---+

      <---URLRW---http://cluster.dominio----->

<----INTRANET-----|                |-----INTERNET----->
```

2. Preparación

Implementé este WebCluster con tres computadores. Obvio que con Linux. =)

La distribución elegida fue RedHat 7.2, aunque es 100% factible realizarla con el sabor favorito de cada uno.

```
PC1 (cluster0): Celeron 600 192 Mb Ram HD 10Gb eth0 (internet) - eth1 (intranet)
PC2 (cluster1): Celeron 400 32 Mb Ram HD 4 Gb eth0 (intranet)
PC3 (cluster2): Celeron 600 256 Mb Ram HD 20Gb eth0 (intranet)
```

Datos técnicos:

```
PC1: eth1 192.168.1.150 [aliased] - eth0: XX.XXX.XXX.XXX (internet)
PC2: eth0 192.168.1.151
PC3: eth0 192.168.1.152
```

[aliased] es debido a que la máquina responde a varios IP de la intranet (192.168.1.1, 192.168.1.2). Otra razón de la buena implementación de TCP-IP en Linux.

PC1 es un servidor WWW/Mail/DNS/Proxy/Router/Firewall.

El aliasing se hizo vía Linuxconf (algunas veces, programas simples sirven bastante para evitar tecleos adicionales).

[**not.ed** esto se puede hacer mediante: `ifconfig eth0:0 num_ip netmask mascara_ip up`] Ah, y no coloqué el IP de internet de PC1 para proteger máquinas inocentes. =)

No hubo compiladas de Kernel ni recompiladas de Apache.

3. Métodos

Los métodos usados fueron dos

3.1 DNS Round Robin

DNS RR es una forma de hacer "rotar" en una "ronda" de números IP que corresponden a un mismo servidor. Para ello se configura el archivo de zona de la siguiente manera:

```
@      IN      SOA  ....
...
...
; zona del cluster

cluster0      IN      A      192.168.1.150
cluster1      IN      A      192.168.1.151
cluster2      IN      A      192.168.1.152

; el cluster
cluster 60    IN      A      192.168.1.150
        60    IN      A      192.168.1.151
        60    IN      A      192.168.1.152

; eof
```

Y configurar Apache a todos los servidores (cluster0-2) de la siguiente manera:

```
# /etc/httpd/conf/httpd.conf
...
...
...
NameVirtualHost <ip>

<VirtualHost <nombre_cluster> >
    DocumentRoot <ruta1>
    CustomLog <ruta_log> vcommon
</VirtualHost>

<VirtualHost cluster.dominio>
    DocumentRoot <ruta2>
    CustomLog <ruta_log> vcommon
</VirtualHost>

# eof
```

Reiniciar Apache y NAMED en todas las máquinas

Reiniciar Apache y Named en todas las máquinas

Al hacer entonces

```
# nslookup cluster.dominio
```

el servidor respondería

```
Name: cluster.dominio
Address: 192.168.1.150
Name: cluster.dominio
Address: 192.168.1.151
Name: cluster.dominio
Address: 192.168.1.152
```

(puede darse un orden distinto, por ejemplo, 192.168.1.152 primero).

Al escribir entonces en el navegador de algún cliente Web de la intranet:

```
http://cluster.dominio
```

Respondería cualquiera de los tres, dependiendo cual este primero en el DNS, sin reescribir el URL en el cliente.

El sistema del DNS Round Robin no es una forma de compartir carga entre servidores, sino una forma de destinar iguales conexiones entre varias máquinas. No hay un balance de carga ni tampoco comparten carga entre ellas. Es el sistema más básico y efectivo.

3.2 URL Rewriting (también llamado Proxy Round Robin)

Este método es totalmente distinto a DNS RR, pero muy similar en su implementación. Por tanto, hay que modificar la zona del dominio, borrar el Round Robin (pero conservar los registros de cluster0, cluster1 y cluster2) y en vez de eso:

```
@ IN SOA .....
```

```
...
```

```
...
```

```
cluster IN A 192.168.1.150
```

O sea, que cluster.dominio apunte solo hacia un número IP.

Reiniciar Named.

No tocar a cluster1 y cluster2 en Apache.

Ahora, configurar a cluster0 (jefe de clusters) en Apache con lo siguiente:

```
# /etc/httpd/conf/httpd.conf
```

```
...
```

```
...
```

```
...
```

```
NameVirtualHost 192.168.1.150
```

```
<VirtualHost cluster0.dominio>
```

```
    DocumentRoot /var/www/cluster0
```

```
    CustomLog /var/log/httpd/cluster0
```

```
</VirtualHost>
```

```
# lo siguiente hace el truco del URL-REWRITE
```

```
<VirtualHost cluster.dominio>
```

```
    RewriteEngine on
```

```
    RewriteMap lb prg:/var/www/balance.pl
```

```
    RewriteRule ^/(.+)$ ${lb:$1} [P,L]
```

```
</VirtualHost>
```

```
# EOF
```

Las opciones [P,L] significan: [P] : (Proxy) Inicializa el modulo de proxy y fuerza una petición [L] : (Last) Detiene el proceso de reescritura de reglas.

Ahora crear un archivo en PERL o cualquier otro lenguaje scripting en la ruta indicada en la línea "RewriteMap lb". En este caso, /var/www

El archivo de ejemplo que coloco es balance.pl que es el siguiente:

```
#!/usr/bin/perl
```

```
# balance.pl - programa de balance de carga
```

```
# hijacked de la documentacion de Apache
```

```
$| = 1;
```

```
$name = "cluster";
```

```
$first = 0;
```

```
$last = 2;
```

```
$domain = "dominio"; # nombre dominio
```

```
$cnt = 0;
```

```
while () {
```

```
    $cnt = (($cnt+1) % ($last+1-$first));
```

```
    $server = sprintf("%s%d.%s", $name, $cnt+$first, $domain);
```

```
    print "http://$server/$_";
```

```
}
```

```
## EOF ##
```

Darle permisos de ejecución (chmod +x balance.pl).

Que es lo que hace este script? Al hacer Enter (al ejecutarlo de la línea de comando) devuelve:

```
# /var/www/balance.pl
```

```
http://cluster0.dominio
```

```
http://cluster1.dominio
```

```
http://cluster2.dominio
```

```
http://cluster1.dominio
http://cluster2.dominio
http://cluster0.dominio
...
```

Que es precisamente lo que se necesita: que Apache reescriba las reglas que van dirigidas a `http://cluster.dominio` hacia `http://cluster<n>.dominio`.

Todo esto gracias a la RewriteRule

ANTES DE REINICIAR APACHE LEER LO SIGUIENTE!

Libproxy no esta configurado por defecto en Apache, asi que hay que agregar un "par" de líneas a `httpd.conf` en `cluster0`

```
...
LoadModule proxy_module      modules/libproxy.so
AddModule mod_proxy.c
<IfModule mod_proxy.c>
    ProxyRequests On
    ProxyVia On
    CacheRoot "/var/cache/httpd"
    CacheSize 5
    CacheGcInterval 4
    CacheMaxExpire 24
    CacheLastModifiedFactor 0.1
    CacheDefaultExpire 1
</IfModule>
...
```

(NOTA: es posible solamente dejar la línea `ProxyVia On`. El resto de las opciones indica la cantidad de páginas que puede hacer cache, tamaño del cache (en Mb), y la fecha de expiración del cache).

Ahora SI reiniciar Apache.

Y probar. =)

4. Ahora, como mostro el resultado a Internet?

Fácil. Sólo cambiar en la zona del dominio ... `cluster IN A 192.168.1.150` ...

por ... `cluster IN A ...`

Reiniciar Named

Y en `cluster0` agregar en `httpd.conf`

```
....
....
NameVirtualHost 192.168.1.150
#agregar lo siguiente
NameVirtualHost
...
```

Y reiniciar Apache, y nada más...

Advertencia: Si el servidor(`cluster0`) esta sirviendo más de un dominio, `<VirtualHost cluster.dominio>` colocarlo al final de `httpd.conf`

5. Agregando más máquinas

Más fácil aún.

Se configura una nueva máquina con el IP (de ejemplo) `192.168.1.153` llamada `cluster3`.

Se agrega un nuevo IP a la zona del dominio:

```
...
cluster3      IN      A      192.168.1.153
...
```

Y configurar Apache en `cluster3` como si fuera otro cluster (`cluster1` o `cluster2`).

Reiniciar Apache en `Cluster3`

Modificar el script `balance.pl` y cambiar

```
...
$last=2;
...
por
...
$last=3;
...
```

6. Y que funciona?

Absolutamente todo. Si todos los nodos del clusters (excepto el principal) fueran (aj) máquinas WIN9X/NT, incluso funcionan los .ASP! Cualquier contenido, PHP, CGIs, SSL, métodos POST-GET-CONNECT. Lo que sea.

7. Algunos cambios adicionales a implementar

7. Algunos cambios adicionales a implementar

Claro, el método es sencillo para tres máquinas, pero se hace igualmente simple para varias máquinas.
Ahora, suponiendo que cluster2 se muere (crash), el sistema dejaría de funcionar cuando balance.pl respondiera <http://cluster2.dominio>.

De ahí que pueda usarse un script externo. Las posibilidades son casi infinitas. =>
Puede usarse en este caso:

- [1] un shellscript
- [2] un programa hecho en C
- [3] un script hecho en Perl

Pero todas estas soluciones deben ser locales para cluster0. En resumen, cluster0 debe funcionar bien.

Si se necesita hacer "una granja de una granja" (es decir, varias máquinas que respondan a internet a cluster.dominio y que los datos sean leídos de una intranet), puede usarse un método de implementación EQUIVALENTE, pero preferentemente para esa solución, es mejor un DNS Round Robin.

Es más claro de explicar con un dibujo:

```

                                +[cluster0]--+
[INTERNET]-->(DNS)[cluster.dominio]-+[cluster0]----[intranet]
                                +[cluster0]--+
```

Esta es una solución implementada por Yahoo y Google.

Volviendo al script, una solución posible podría ser la siguiente:

- [1] Determinar que cluster es el que respondería
- [2] Hacer una conexión HTTP haciendo un GET alive.html (el servidor debería responder con alguna página o con un error. lo que hice fue que respondiera con un "i'm alive") con un timeout de 2 segundos como mínimo
- [3] Si no responde, volver al paso [1]
- [4] Devolver el cluster correspondiente

8. Otras cosas posibles con mod_rewrite

Según Brian Moore, "mod_rewrite es vudu" =>. Hay muchas cosas especiales que pueden hacerse gracias a mod_rewrite. Incluso cosas que pueden hacerse vía configuraciones de Apache, como controles de acceso. Es sólo cosa de combinar un par de reglas y un resultado.

Es necesario para esto dominar

- [1] Expresiones regulares
- [2] RewriteRule y RewriteMap de mod_rewrite.

Entonces, no hay como un buen RTFM.

Algunas indicaciones de RewriteRule:

- [NC] : Las peticiones se vuelven NO sentitivas a las mayusculas y minusculas
- [NE] : No escape: evita que Apache reescriba, por ejemplo, el caracter tilde (~) por %7E
- [F] : Forbidden. Automaticamente genera un error 403
- [G] : Gone. Fuerza un error 410.

Recordar algo adicional ademas.

Hay formas de reescribir el URL "per-server" o "per-directorio".

"Per-server" son reescrituras de URI hechas en httpd.conf

"Per-directorio" son reescrituras hechas en los archivos de configuracion por directorio (.htaccess)

9. Documentación

Todo esta implementacion de un Webcluster se puede encontrar en URL Rewriting Guide de Apache. Claro que esta fue la manera lenta de hacer un Webcluster.

