

Load Balancer



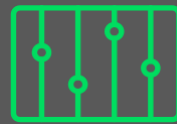
Content Cache



Web Server



Security Controls



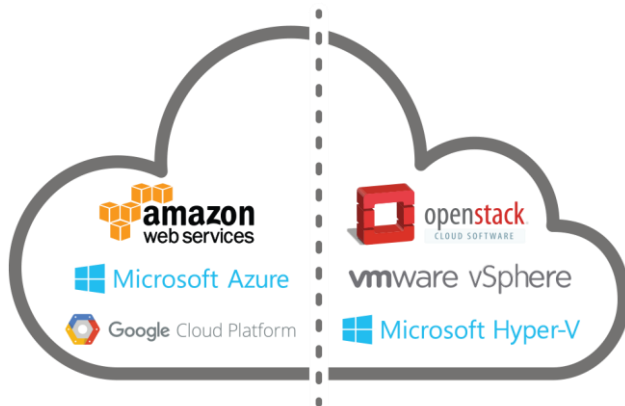
Monitoring &
Management

Impecable entrega de aplicaciones para la
Web moderna

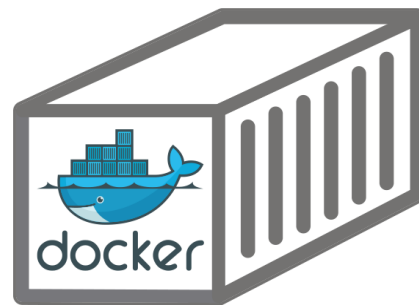
Funcionan sobre cualquier entorno



Bare Metal



Public/Private/Hybrid Cloud



Containers



PROXYING CONNECTIONS

NGINX

NGINX – Proxy

- Configuración de Proxy Server
- Uso de la directive “proxy_pass”
- Uso de Proxy Buffering

Reverse Proxy Server

- Nginx, soporta proxy para HTTP, HTTPS, TCP, UDP, y otros protocolos.

```
server {  
    listen 80;  
    server_name example.com;  
  
    location / {  
        proxy_pass http://tomcat-backend;  
    }  
}
```

Directiva proxy_pass

- Definir la dirección y protocolo del Proxy Server.

Syntax:

```
proxy_pass <destination>
```

Example:

```
location / {  
    proxy_pass http://backend:8080/application/  
}
```

Proxy_pass con un Path

- Definir la dirección request URI y protocolo del Proxy Server.

location prefix:

```
#curl request example:
$ curl http://server/application1/

#NGINX selects a matching location prefix
location /application1 {
    proxy_pass http://localhost:8080/otherapp;
}

#proxy request becomes
http://server:8080/otherapp/index.html;
```


Proxy_pass sin un Path

- Definir la dirección request URI y protocolo del Proxy Server.

```
#curl request example:  
$ curl http://server/application2/  
  
#NGINX selects a matching location prefix  
location /application2 {  
    proxy_pass http://localhost:8080  
}
```


Lab 1. Configuración Proxy Reverse

1. Crear una nueva configuración `server2.conf`

```
$ sudo vim /etc/nginx/conf.d/server2.conf
```

2. Definir el siguiente context `server`, luego guardar el archive.

```
server {  
    listen 8080;  
    root /data/server2;  
}
```

Lab 1. Configuración Proxy Reverse

3. Abrir el archivo `server1.conf`

```
$ sudo vim /etc/nginx/conf.d/server1.conf
```

4. Adicionar un `proxy_pass`, en `/application1` con la URI:

```
location /application1 {  
    proxy_pass http://localhost:8080/sampleApp;  
}
```

5. Guardar y reiniciar el servicio Nginx.

6. Validar con el commando curl o navegador

<http://IP/application1>, que se muestra?



LOAD BALANCING

NGINX

NGINX – Objetivos

- Configurar Load Balancer
- Identificar los métodos de Balanceo
- Otras Configuraciones

Load Balancing

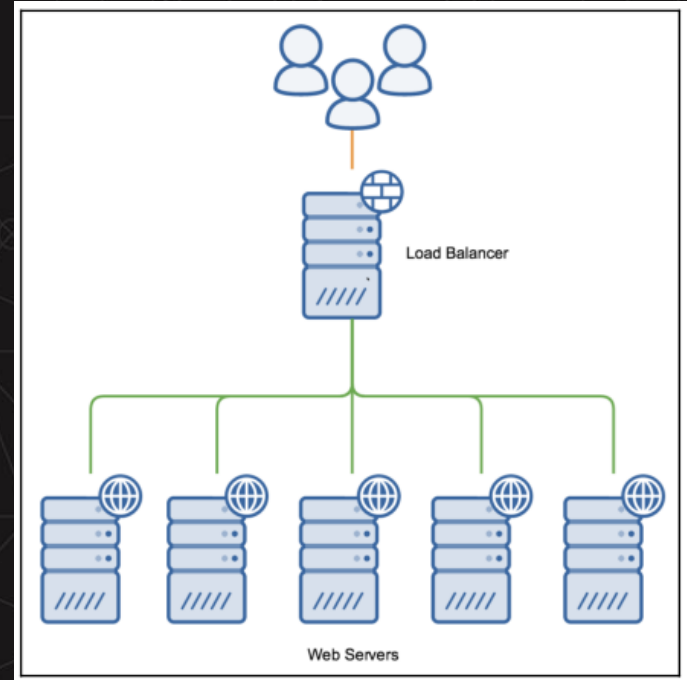
RESUMEN:

El equilibrio de carga en varias instancias de aplicaciones es una técnica comúnmente utilizada para optimizar el uso de recursos, maximizar el rendimiento, reducir la latencia y garantizar configuraciones tolerantes a fallas.

NGINX y NGINX Plus se pueden usar en diferentes escenarios de implementación como un equilibrador de carga HTTP muy eficiente

Enviando tráfico HTTP a un Grupo de Servidores

- Para comenzar a utilizar NGINX Plus o NGINX para equilibrar la carga de tráfico HTTP a un grupo de servidores, primero debe definir el grupo con la directiva “**upstream**”, la directiva se coloca en contexto “**http**”.



Enviando tráfico HTTP a un Grupo de Servidores

Los servidores del grupo se configuran utilizando la directiva de “**server**” (que no debe confundirse con el bloque de “**server**”, que define un servidor virtual que se ejecuta en NGINX). Por ejemplo, la siguiente configuración define un grupo llamado “**backend**” y consta de tres configuraciones de servidor (que pueden resolverse en más de tres servidores reales):

```
http {  
    upstream backend {  
        server backend1.example.com weight=5;  
        server backend2.example.com;  
        server 192.0.0.1 backup;  
    }  
}
```


Enviando tráfico HTTP a un Grupo de Servidores

Para pasar solicitudes a un grupo de servidores, el nombre del grupo se especifica en la directiva “`proxy_pass`” (o las directivas `fastcgi_pass`, `memcached_pass`, `scgi_pass` o `uwsgi_pass`, para esos protocolos). En el siguiente ejemplo, un servidor virtual que se ejecuta en NGINX pasa todas las solicitudes al grupo upstream “`backend`” definido en el ejemplo anterior:

```
server {  
    location / {  
        proxy_pass http://backend;  
    }  
}
```

Enviando tráfico HTTP a un Grupo de Servidores

- El siguiente ejemplo combina los dos fragmentos para enviar solicitudes HTTP al grupo de servidores “backend”
- El grupo consta de tres servidores, dos de ellos ejecutan instancias de la misma aplicación, mientras que el tercero es un servidor de respaldo. Debido a que no se especifica ningún algoritmo de balanceo de carga.
- NGINX usa el algoritmo predeterminado, Round Robin:

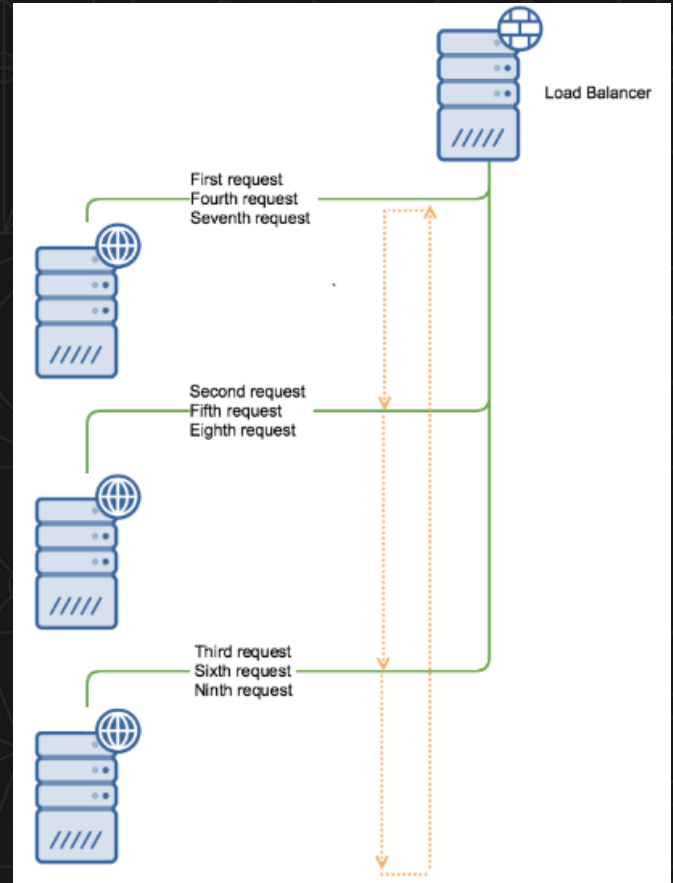
```
http {  
    upstream backend {  
        server backend1.example.com;  
        server backend2.example.com;  
        server 192.0.0.1 backup;  
    }  
  
    server {  
        location / {  
            proxy_pass http://backend;  
        }  
    }  
}
```

Métodos de Balanceo

- Los tres algoritmos de programación compatibles con NGINX son **round-robin**, **least-connections** y **hashing**.

- ROUND-ROBIN**

Los equilibradores de carga configurados en **round-robin** distribuyen las solicitudes entre los servidores de forma secuencial; la primera solicitud va al primer servidor, la segunda solicitud al segundo servidor, y así sucesivamente. Esto se repite hasta que cada servidor del grupo haya procesado una solicitud.



Métodos de Balanceo

LEAST CONNECTIONS

- Se envía una solicitud al servidor con la menor cantidad de conexiones activas, una vez más teniendo en cuenta los pesos del servidor:

```
upstream backend {  
    least_conn;  
    server backend1.example.com;  
    server backend2.example.com;  
}
```

Métodos de Balanceo

IP HASH

- El servidor al que se envía una solicitud se determina a partir de la dirección IP del cliente. En este caso, los tres primeros octetos de la dirección IPv4 o toda la dirección IPv6 se utilizan para calcular el valor hash. El método garantiza que las solicitudes de la misma dirección lleguen al mismo servidor a menos que no esté disponible.

```
upstream backend {  
    ip_hash;  
    server backend1.example.com;  
    server backend2.example.com;  
}
```


Métodos de Balanceo

- Si uno de los servidores necesita ser eliminado temporalmente de la rotación de balanceo de carga, se puede marcar con el parámetro “down” para preservar el hashing actual de las direcciones de IP del cliente. Las solicitudes que debía procesar este servidor se envían automáticamente al siguiente servidor del grupo:

```
upstream backend {  
    server backend1.example.com;  
    server backend2.example.com;  
    server backend3.example.com down;  
}
```

Server Weights

- Por defecto, NGINX distribuye las solicitudes entre los servidores en el grupo de acuerdo con sus pesos utilizando el método **Round Robin**. El parámetro de peso de la directiva del servidor establece el peso de un servidor; el valor predeterminado es 1:

```
upstream backend {  
    server backend1.example.com weight=5;  
    server backend2.example.com;  
    server 192.0.0.1 backup;  
}
```

backend1.example.com tiene un peso de 5; los otros dos servidores tienen el peso predeterminado (1), pero el que tiene la dirección IP 192.0.0.1 está marcado como “backup” y no recibe solicitudes a menos que los otros servidores no estén disponibles.

Con esta configuración de pesos, de cada seis solicitudes, cinco se envían a backend1.example.com y una a backend2.ejemplo.com.

Server Slow-Start

- La función `slow-start` evita que un servidor recientemente recuperado se vea abrumado por las conexiones, lo que puede agotar el tiempo de espera y provocar que el servidor se marque como fallido nuevamente.

```
upstream backend {  
    server backend1.example.com slow_start=30s;  
    server backend2.example.com;  
    server 192.0.0.1 backup;  
}
```

El valor de tiempo (en este caso, 30 segundos) establece el tiempo durante el cual NGINX Plus incrementa el número de conexiones al servidor al valor completo.

Persistencia de Session

- La persistencia de la sesión significa que NGINX Plus identifica las sesiones de usuario y enruta todas los **request** de sesión al mismo **server upstream**.

Sticky Cookie

- NGINX Plus agrega una cookie de sesión a la primera respuesta del grupo **upstream** e identifica el servidor que envió la respuesta. La siguiente solicitud del cliente contiene el valor de la cookie y NGINX Plus enruta la solicitud al **server upstream** que respondió a la primera solicitud:

```
upstream backend {  
    server backend1.example.com;  
    server backend2.example.com;  
    sticky cookie srv_id expires=1h domain=.example.com path=/  
}
```

srv_id -- nombre de la cookie
1h – El navegador guarda cookie
Domain – dominio
/ -- Path

Limitar Número de Conexiones

- Con NGINX Plus, es posible limitar el número de conexiones a un **server upstream** especificando el número máximo con el parámetro **max_conns**.
- Si se alcanza el límite de **max_conns**, la solicitud se coloca en una cola para su posterior procesamiento, siempre que la **directiva queue** se incluya para establecer el número máximo de solicitudes que pueden estar simultáneamente en la cola:

```
upstream backend {  
    server backend1.example.com max_conns=3;  
    server backend2.example.com;  
    queue 100 timeout=70;  
}
```

Si la cola se llena con solicitudes o el **server upstream** no se puede seleccionar durante el tiempo de espera del **timeout** especificado, el cliente recibe un error.

Passive Health Checks

- Cuando NGINX considera que un servidor no está disponible, deja de enviar solicitudes al servidor temporalmente hasta que se considere activo nuevamente. Los siguientes parámetros de la directiva del servidor configuran las condiciones bajo las cuales NGINX considera que un servidor no está disponible:

max_fails - Establece el número de intentos fallidos consecutivos después de los cuales NGINX marca el servidor como no disponible.

fail_timeout - establece el tiempo durante el cual la cantidad de intentos fallidos especificados por el parámetro **max_fails** debe ocurrir para que el servidor se considere no disponible, y también el período de tiempo que NGINX considera que el servidor no está disponible después de que se marcó.

```
upstream backend {  
    server backend1.example.com;  
    server backend2.example.com max_fails=3 fail_timeout=30s;  
    server backend3.example.com max_fails=2;  
}
```

Active Health Checks

- A continuación se presentan algunas características más sofisticadas para rastrear la disponibilidad del servidor en NGINX Plus.
- Periódicamente se envían solicitudes especiales a cada servidor para validar una respuesta que satisfaga ciertas condiciones puede monitorear la disponibilidad de los servidores, para lo cual es necesario habilitar en su archivo `nginx.conf`, la directiva `health_check`

```
http {  
    upstream backend {  
        zone backend 64k;  
        server backend1.example.com;  
        server backend2.example.com;  
        server backend3.example.com;  
        server backend4.example.com;  
    }  
    server {  
        location / {  
            proxy_pass http://backend;  
            health_check;  
        }  
    }  
}
```


Lab 2. Balanceo Nginx

1. En el directorio `/etc/nginx/conf.d/` crear un Nuevo archivo llamado “`backends.conf`” y definir tres contextos “`server`” con el valor de `root` a `/data/backend1`, `backend2` y `backend3` respectivamente y debe escuchar en los puertos `8081`, `8082` y `8083`

Ejemplo:

```
server {  
    listen 8081;  
    root /data/backend1;  
}  
server {  
    listen 8082;  
    root /data/backend2;  
}  
...
```

Lab 2. Balanceo Nginx

2. Validar la configuración realizada con el commando curl:

Ejemplo:

```
# curl http://localhost:8081
```

3. Backup **server1.conf** y crear un archivo llamado **myServers.conf** y definir un **upstream** con tres **servers** usando **127.0.0.1:8081**, **127.0.0.1:8082** y **127.0.0.1:8083**.

4. Crear un context **server** con el valor de listen en 8080. Setear la directiva **root** al directorio **/var/public_html**

5. Definir un **error_log** con el nivel **info** y un **access_log** con el nivel **combined**.

6. Crear un context **location** igual a **/**

7. Adicionar un **proxy_pass** que reenvíe todos los request al **upstream** creado.

Lab 2. Balanceo Nginx

```
[root@serverins conf.d]# cat backends.conf
server {
    listen 8081;
    root /data/backend1;
}
server {
    listen 8082;
    root /data/backend2;
}
server {
    listen 8083;
    root /data/backend3;
}
```

Lab 2. Balanceo Nginx

```
[root@serverins conf.d]# cat myServers.conf
upstream myServers {
    server 127.0.0.1:8081; #backend1
    server 127.0.0.1:8082; #backend2
    server 127.0.0.1:8083; #backend3
}

log_format test_log "Request: $request \nStatus: $status \nUpstream IP: $upstream_addr \n";

server {
    listen 8080;
    root /var/public_html;
    error_log /var/log/nginx/upstream.error.log info;
    access_log /var/log/nginx/upstream.access.log test_log;

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

Lab 2. Balanceo Nginx

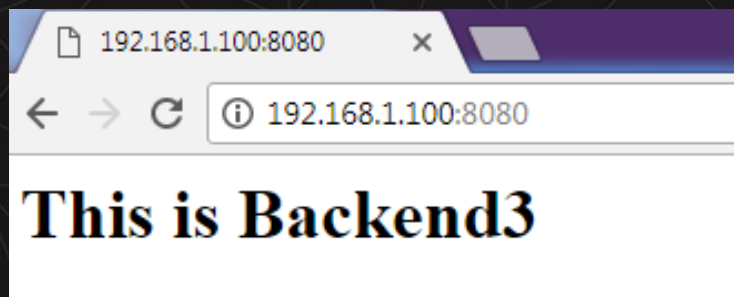
1. Validar el funcionamiento del balanceo:

```
[root@serverins conf.d]# curl http://localhost:8080
<h1>This is Backend3</h1>
[root@serverins conf.d]# curl http://localhost:8080
<h1>This is Backend1</h1>
[root@serverins conf.d]# curl http://localhost:8080
<h1>This is Backend2</h1>
[root@serverins conf.d]# curl http://localhost:8080
<h1>This is Backend3</h1>
[root@serverins conf.d]# curl http://localhost:8080
<h1>This is Backend1</h1>
[root@serverins conf.d]# curl http://localhost:8080
<h1>This is Backend2</h1>
[root@serverins conf.d]# curl http://localhost:8080
<h1>This is Backend3</h1>
```

Lab 2. Balanceo Nginx

1. Validar el funcionamiento del balanceo:

```
[root@serverins conf.d]# curl http://localhost:8080
<h1>This is Backend3</h1>
[root@serverins conf.d]# curl http://localhost:8080
<h1>This is Backend1</h1>
[root@serverins conf.d]# curl http://localhost:8080
<h1>This is Backend2</h1>
[root@serverins conf.d]# curl http://localhost:8080
<h1>This is Backend3</h1>
[root@serverins conf.d]# curl http://localhost:8080
<h1>This is Backend1</h1>
[root@serverins conf.d]# curl http://localhost:8080
<h1>This is Backend2</h1>
[root@serverins conf.d]# curl http://localhost:8080
<h1>This is Backend3</h1>
```



Lab 2. Balanceo Nginx

2. Validar los logs configurados

```
[root@serverins conf.d]# tail -f /var/log/nginx/upstream.access.log
Upstream IP: 127.0.0.1:8081

Request: GET / HTTP/1.1
Status: 200
Upstream IP: 127.0.0.1:8082

Request: GET / HTTP/1.1
Status: 200
Upstream IP: 127.0.0.1:8083
```

Lab 3. Monitoreo Balanceo

1. En el archivo `backends.conf` especificar el server zone para cada server usando la directiva `status_zone`.

Ejemplo:

```
server {  
    listen 8081;  
    root /data/backend1;  
    status_zone backend1;  
}
```

...

2. Guardar y reiniciar el servidor Nginx, luego validar el monitoreo validando el balanceo de carga..

Lab 3. Monitoreo Balanceo

3. Abrir el archivo `myServers.conf` y especificar un shared memory zone en el bloque `upstream` de 64k.

zone http_backend 64k;

2. Guardar y reiniciar el servidor Nginx, luego validar el monitoreo validando el balanceo de carga y los valores asignados

Lab 4. Modo Mantenimiento

1. Abrir el archivo `myServers.conf` y adicionar un bloque match con el nombre `health_condition` que valida el code status, header response y body response.

```
match health_conditions {  
    status 200-399;  
    header Content-Type = text/html;  
    body !~ mantenimiento;  
}
```

2. En el prefijo `/` adicionar el parametro `health_check`

```
location / {  
    proxy_pass http://myServers;  
    health_check match=health_conditions  
    fails=2  
    uri=/health/test.html;  
}
```

Lab 4. Modo Mantenimiento

3. La configuración del archivo myServers.conf sería:

```
[root@serverins conf.d]# cat myServers.conf
upstream myServers {
    zone http_backend 64K;
    server 127.0.0.1:8081; #backend1
    server 127.0.0.1:8082; #backend2
    server 127.0.0.1:8083; #backend3
}

log_format test_log "Request: $request \nStatus: $status \nUpstream IP: $upstream_addr \n";

server {
    listen 8080;
    root /var/public_html;
    error_log /var/log/nginx/upstream.error.log info;
    access_log /var/log/nginx/upstream.access.log test_log;


    location / {
        proxy_pass http://myServers;
        health_check match=health_conditions fails=2 passes=3 uri=/health/test.html;
    }
}

match health_conditions {
    status 200-399;
    header Content-Type = text/html;
    body !~ mantenimiento;
}
```

Lab 4. Modo Mantenimiento

3. Modificar el contenido de página html: `/data/backend3/health/test.html`, con la palabra: **mantenimiento** y validar el monitoreo.

Regresar el valor del contenido de la página `test.html`



Upstreams [Show upstreams list](#)

myServers Zone: **40%**

Server			Requests	
Name	DT	W	Total	Req/s
127.0.0.1:8083	1m	1	0	0
127.0.0.1:8081	1m	1	0	0
127.0.0.1:8082	1m	1	0	0

Questions and Next Steps

NGINX