Load Balancing and Failover with an NGINX Server at the Edge and an IPv6-Only Web Server Farm

Author: Alejandro Acosta
Coordination and Revision: Guillermo Cicileo, Carlos MArtínez
Edition: Communications Area
Department: Technology Area

August 2023

lacnic
www.lacnic.net

Introduction

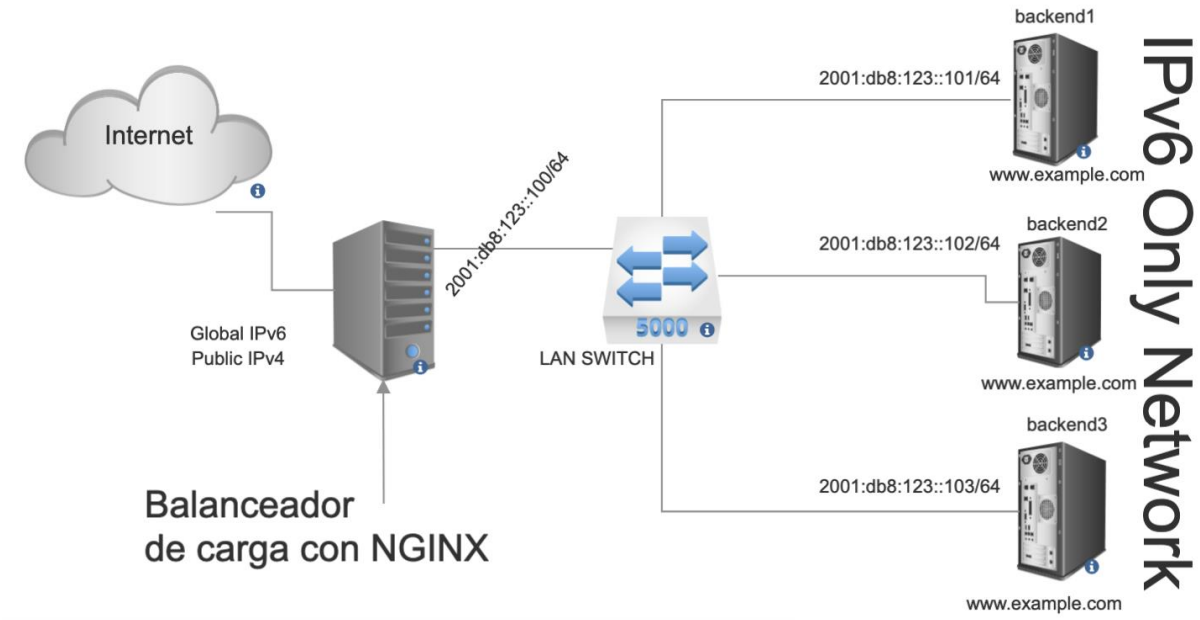
This paper is the continuation of NGINX Reverse Proxy for an IPv6-Only Server Farm: Efficient Web Connectivity. In that document we showed how to configure an NGINX Reverse Proxy and, together with a server, we were able to provide dual-stack (IPv4 and IPv6) web access to an IPv6-only server farm. A very interesting way to save IPv4 addresses and obtain a number of additional benefits.

In this article, we will explore how to implement load balancing capabilities using an NGINX server at the edge and a web server farm operating exclusively on IPv6. We will describe the benefits of this configuration and the steps necessary to achieve a robust and reliable architecture, as well as the different implementation methods.

Why use NGINX at the edge?

NGINX servers are known for their performance, scalability, and advanced load balancing capabilities. Placing an NGINX server at the edge of the network allows you to have a single point of entry for your web services, where you can efficiently manage and distribute traffic to your IPv6-only server farm.

Topology



NGINX load balancing methods

NGINX has several methods for load balancing, each of which is explained below.

- IP Hash: This method uses an algorithm that takes the source and destination IP addresses of the client and server and generates a unique hash key. This allows session persistence.
- Round Robin (default): This is the default load balancing method. It tells the load balancer to go back to the top of the list and repeat the process.
- Least Connections (least_conn): This method uses a dynamic load balancing algorithm. It redistributes connections to the member of the pool with the least number of active connections when a new connection request is received.

NGINX load balancing configuration requirements (proxy server -edge-)

- Install NGINX on your edge server and make sure it is properly configured to work with both IPv4 and IPv6. Keep in mind that this server can listen on both IPv4 and IPv6, it will proxy the requests and forward them internally to the server farm via IPv6 only.
- Create an NGINX configuration file and define the upstream block with the IPv6 addresses of your web servers.
- Configure the load balancing algorithms (e.g., round-robin, least_conn, or ip_hash) to distribute requests among the web servers in the farm.
- Linux server at the edge with NGINX installed and with an IPv4 address and an IPv6 address.
- Each of the web servers that are part of your farm must have a different IPv6 address configured, and these IP addresses must be reachable by the proxy server.

Configurations

Load balancer side configuration:

```
#File: /etc/nginx/sites-enabled/example.com
upstream backend {  #The upstream of the server farm is called upstream
    server [2001:db8:123::101]; #server backend1
    server [2001:db8:123::102]; #server backend2
    server [2001:db8:123::103]; #server backend3
}

server {  #this is a known NGINX directive
    listen 80; #port on which the web server listens
    server_name example.com www.example.com;  #domain name

    location / {
        proxy_pass http://backend;   #note that backend is the name of the
upstream
    }
}
```

Server farm side configuration

All backend servers in the farm have the same configuration.

```
#/etc/nginx/sites-available/default
server {
  listen [::]:80 default_server;
  root /var/www/html;
  index index.html index.htm index.nginx-debian.html;
  server_name _;
  location / {
    try_files $uri $uri/ =404;
  }
}
```

Testing and monitoring

Once the configurations are complete, we can proceed with testing. Here is a list of tests that can potentially be performed:

a) Create a different file on each server in the farm and load www.example.com from the Internet. Every time we load and reload the page, it should show a page for each server in the upstream directive.
b) Check the nginx logs on each backend server, e.g., you could use *tail -f /var/log/nginx/*.log* and check all accesses and/or errors.
c) Review the load balancer logs, you could also use *tail -f /var/log/nginx/*.log*
d) You could run this very simple script to test the round robin:
for ((i=1;i<=10;i++)); do curl -v "http://www.example.com"; sleep 1; done

NGINX configuration for failover and other options

In NGINX, failover is handled by sending parameters to the specified upstream backend servers.

The various parameters are as follows (see example below):

weight: This option allows you to specify the relative weight of each server in the upstream group. As you have already used it in your example, the weight determines the proportion of requests that each server will handle compared to the others.

max_fails: This option allows you to specify the maximum number of failed attempts to connect to a server before considering it temporarily unavailable. The default value is 1. For example, max_fails=3; specifies that a server will be marked as unavailable after three consecutive failed connection attempts.

fail_timeout: This option defines the time that a server will be considered unavailable after reaching the maximum number of failed attempts specified by max_fails. The default value is 10 seconds. For example, fail_timeout=30s; defines a 30 second timeout for a server marked as unavailable.

backup: This option specifies that a server should be used as a reserve or backup. A server marked as backup will only be used if all other servers are marked as unavailable.

down: This option marks a server as permanently unavailable. NGINX will not send requests to a server marked as “down” even if all other servers are marked as unavailable. For example, down; marks a server as unavailable.

```
upstream backend {
    server [2001:db8:123::101] weight=3;
    server [2001:db8:123::102] max_fails=2 fail_timeout=10s;
    server [2001:db8:123::103] backup;
    server [2001:db8:123::104] down;
}
```

Check configuration and restart the server to apply the changes

```
# nginx -t
# systemctl restart nginx
```

Conclusion

Implementing load balancing and failover using an NGINX server at the edge and an IPv6-only web server farm results in a scalable and robust architecture. You can efficiently distribute incoming traffic between your web servers and ensure high availability for your services.

GitHub with the configuration files

https://github.com/LACNIC/BlogPostHelpFiles/tree/main/2023_07_Balanceo_Carga_y_Failover_NGINX_IPv6

References

https://help.clouding.io/hc/es/articles/360019908839-C%C3%B3mo-configurar-un-servidor-de-balanceo-de-carga-Nginx-en-Ubuntu-20-04
https://cloud.google.com/load-balancing/docs/https
https://stackoverflow.com/questions/69285690/nginx-load-balancer-configuration-not-working