NGINX Reverse Proxy for an IPv6-Only Server Farm: Efficient Web Connectivity

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

July 2023

lacnic
www.lacnic.net

Introduction

This work presents a very simple way to offer dual-stack web access to an IPv6-only server farm using NGINX. The continued growth of the Internet and the gradual adoption of the IPv6 protocol means that it is essential to ensure connectivity and accessibility for clients using both IPv4 and IPv6.

We will explain how to configure NGINX to support dual-stack web access, we will address how to configure NGINX as a reverse proxy that listens on both IPv4 and IPv6 addresses, as well as how to correctly route incoming requests to backend servers with only IPv6 addresses. By the way, among many other benefits, what we will discuss in the following article is an important step towards the preservation of IPv4 addresses.

What is a reverse proxy?

In [1], Cloudflare defines a Reverse Proxy Server as follows:

“A reverse proxy is a server that sits in front of web servers and forwards client (e.g. web browser) requests to those web servers. Reverse proxies are typically implemented to help increase security, performance, and reliability. In order to better understand how a reverse proxy works and the benefits it can provide, let’s first define what a proxy server is.”

What is a proxy server?

In [1], Cloudflare also provides the following definition for a proxy server:

“A forward proxy, often called a proxy, proxy server, or web proxy, is a server that sits in front of a group of client machines. When those computers make requests to sites and services on the Internet, the proxy server intercepts those requests and then communicates with web servers on behalf of those clients, like a middleman.”

What are the benefits of a reverse proxy?

- A reverse proxy can offer IPv4 or transparent IPv6 to clients serviced from an IPv6-only server farm (which is what we will focus on).

- **Scalability:** The use of a reverse proxy allows adding or removing backend servers as needed without affecting end users. This makes it easier for applications to scale out, allowing them to handle a larger number of concurrent users and requests.

- **Static content caching:** NGINX can cache static content such as images, CSS files, and JavaScript, thus reducing the load on backend servers and increasing content delivery speed. This decreases page load times and the required bandwidth.

- **Security:** NGINX acts as a point of entry to the application, providing an additional layer of security. It can perform functions such as request filtering, DDoS attack prevention, SQL injection protection, and client authentication. NGINX can also enable the use of SSL/TLS encryption for communication between clients and the backend server.

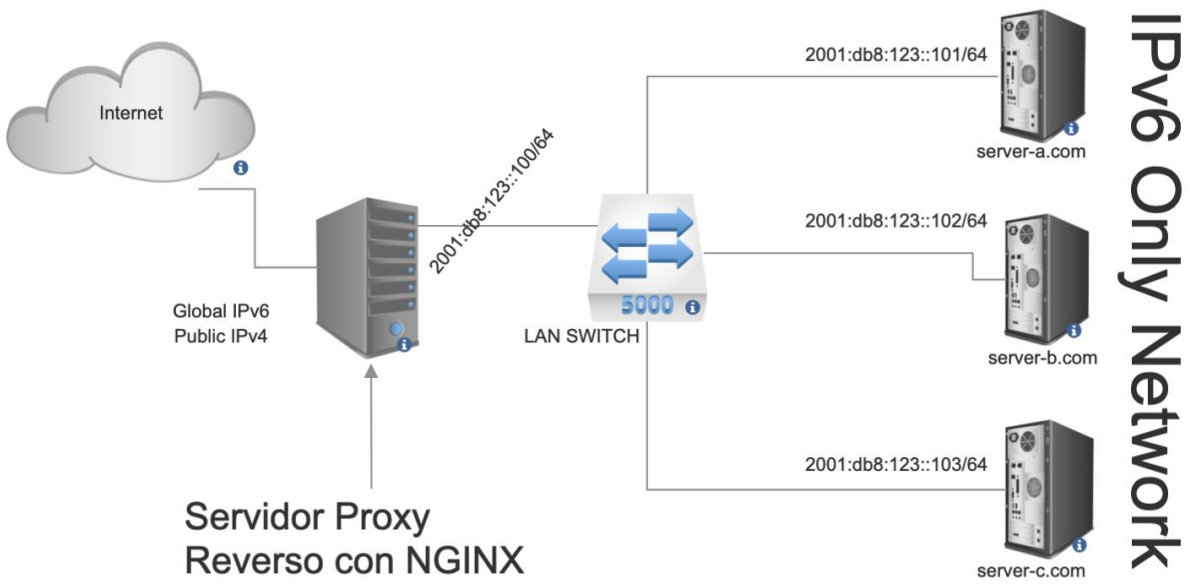
- **Advanced routing:** A reverse proxy allows performing advanced routing based on various criteria, such as domain name, URL, or HTTP headers. This is useful when we need to direct traffic to different backend servers based on the specific attributes of the requests.

- **Consolidation of services:** NGINX can act as a single point of entry for various backend services. This simplifies the infrastructure by consolidating multiple services on a single server, thus simplifying management and maintenance.

- **Enhanced performance**: NGINX is lightweight and resource efficient by design. Its streamlined architecture and ability to handle large numbers of concurrent connections make it a popular choice for improving web app performance.

- **Load balancing:** A reverse proxy such as NGINX can distribute incoming traffic across several backend servers. This helps balance the load and guarantees that no server is overloaded, which improves an application's performance and responsiveness.

Topology to be used



What is our goal for today?

The edge server (Reverse Proxy Server) will be able to receive IPv4 and IPv6 HTTP requests, and depending on the website a user wishes to visit (domain), will forward the request to the right server. This is what will happen in our example:

The client visits:		The request is sent to:
server-a.com	→	2001:db8:123::1
server-b.com	→	2001:db8:123::2
server-c.com	→	2001:db8:123::3
server-a.com	→	2001:db8:123::101
server-b.com	→	2001:db8:123::102
server-c.com	→	2001:db8:123::103

Requirements

- Linux with NGINX on the Reverse Proxy Server
- Super user access
- Web server on each of the servers in the farm
- IPv4 and IPv6 Internet connectivity
- Internal IPv6 connectivity

Let's get started

1) Install NGINX in all servers

```
#apt update
#apt install nginx
```

2) Create the websites in the NGINX reverse proxy

File /etc/nginx/sites-available/server-a.com

```
server {
listen 80;
listen [::]:80;

server_name server-a.com;
location / {
proxy_pass http://[2001:db8:123::101];
}
}
```

File /etc/nginx/sites-available/server-b.com

```
server {
listen 80;
listen [::]:80;

server_name server-b.com;
location / {
proxy_pass http://[2001:db8:123::102];
}
}
```

File /etc/nginx/sites-available/server-c.com

```
server {
listen 80;
listen [::]:80;

server_name server-c.com;
location / {
proxy_pass http://[2001:db8:123::103];
}
}
```

3) Create symbolic links to enable the sites configured above:

root@ProxyReverseSRV:/etc/nginx/sites-enabled# ln -s /etc/nginx/sites-available/server-a.com /etc/nginx/sites-enabled/server-a.com

root@ProxyReverseSRV:/etc/nginx/sites-enabled# ln -s /etc/nginx/sites-available/server-b.com /etc/nginx/sites-enabled/server-b.com

root@ProxyReverseSRV:/etc/nginx/sites-enabled# ln -s /etc/nginx/sites-available/server-c.com /etc/nginx/sites-enabled/server-c.com

4) Remember to restart NGINX:

$sudo systemctl restart nginx

About the logs

Logs are extremely important for any company or ISP that wishes to review incoming connections.

By default, NGINX will use its own IP address for outgoing connections, which results in the loss of the address of the client that originated the HTTP request. But don't worry. NGINX has the solution: *proxy_set_header*. This requires configuring both the end server and the Reverse Proxy server.

1) On the Reverse Proxy Server, we must configure the website assets.

```
# Example of nginx reverse proxy that allows logging the
client's
# original address and port number
location /examples {
  proxy_pass http://[2001:db8:123::103];
  proxy_buffering off;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-Host $host;
  proxy_set_header X-Forwarded-Port $server_port;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}
```

2) On the end server, add the following in the http section of the /etc/nginx/nginx.conf file:

```
   set_real_ip_from 2001:db8:123::100; #replace the IP address
with that of the proxy
   real_ip_header X-Forwarded-For;
   real_ip_recursive on;
```

Example:

```
http {
    …
    set_real_ip_from 2001:db8:123::100;
    real_ip_header X-Forwarded-For;
    real_ip_recursive on;
   …
}
```

With these settings, the receiving server will trust the X-Forwarded-For header set to 2001:db8:123::100 and will log the client's source IP to /var/log/nginx/access.log.

Conclusions

The proposed design allows managing a 100% IPv6-only web server farm with access to both the IPv4 and the IPv6 worlds in a very simple, scalable, and efficient manner. This results in various benefits, including having to manage only one TCP/IP stack, simplicity, security, and even saving IPv4 addresses.

References

- [1] https://www.cloudflare.com/es-es/learning/cdn/glossary/reverse-proxy/
- https://www.digitalocean.com/community/tutorials/how-to-configure-nginx-as-a-reverse-proxy-on-ubuntu-22-04
- GitHub. LACNIC Blog Post Help Files for the entire project: https://github.com/LACNIC/BlogPostHelpFiles/tree/main/2023_Ofreciendo_conectividad_Dual_Stack_a_servidores_Web_en_una_granja_de_servidores_100_IPv6_Only