AUGUST 2025

HOMELAB: REARCHITECTING MY NETWORK AFTER AN ISP CHANGE

BLAKE NIEBRUGGE
SECURITY ANALYST

Table of Contents

Abstract

This case study describes how I re-engineered my homelab networking environment after changing Internet Service Providers (ISPs) and losing a publicly routable IPv4 due to Carrier-Grade Network Address Translation (CGNAT). The change broke how I hosted my websites and my VPN – forcing me to look into more modern solutions. I implemented Cloudflare Tunnel to restore access to the websites and Tailscale to poke through the new restrictions and re-establish private VPN-like connectivity.

If you have any questions/comments/concerns about the content in this case, please let me know using the email address below to reach me.

Thank you,

Blake Niebrugge
Security Analyst
blakeniebrugge.com
blake@blakeniebrugge.com

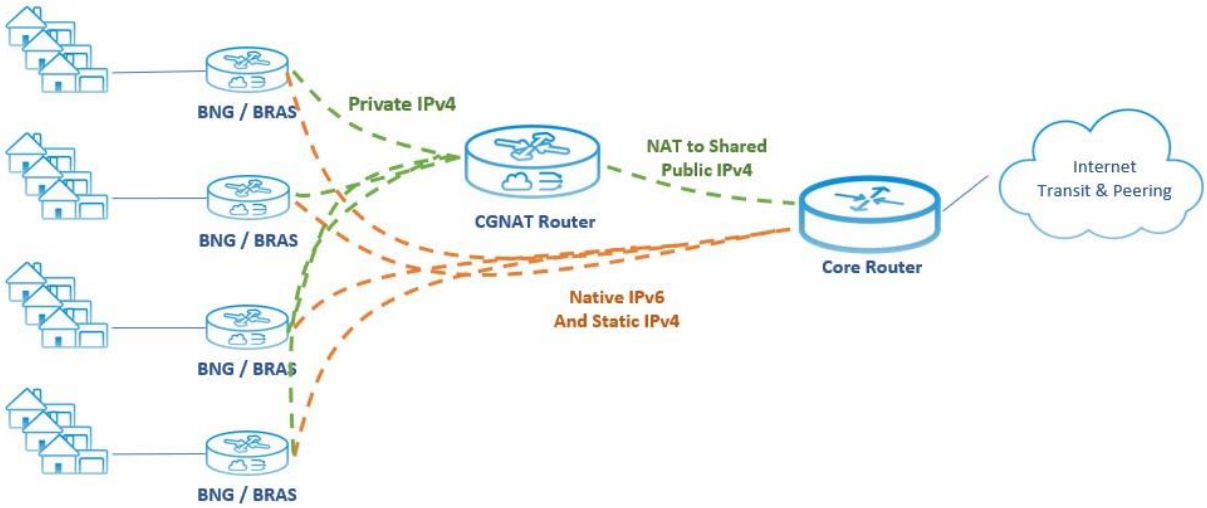
Part 1: The Problem

As mentioned in the abstract, this was a real world case that I had to resolve and figured it would be beneficial to myself to document what the problem was, the approach was to resolve, and how I actually resolved it. So let me set up the problem:

As some may know, I run a personal homelab on my home network, which is really just a Dell Optiplex 3040 MMF that runs a Proxmox virtual environment. I host a number of things in this environment, which includes a web server and a VPN server that have to have access from the internet. Originally with my first internet provider, this was easy to do. I got a public IP address (which could change from time to time), I forwarded the ports I needed to my router, and it was accessible. I was able to navigate the possibly changing IP addresses by utilizing a Dynamic DNS script on my servers – which essentially was periodically checking my public IP address I was assigned, and if there was any changes, it made an API call to my DNS provider to update the A records for respective subdomains.

I made a change to fiber internet and immediately ran into issues with my hosting. I updated my IP addresses to my new ISP and forwarded the ports as I had done before, but they were not returning anything. I noticed that my public IP listed on my router and what was listed when I searched my IP address were different. This is when I realized that my new ISP was utilizing a CGNAT or Carrier-Grade Network Address Translation in their architecture. I had never heard of this before, but smaller ISPs will do this to save on the number of public IPv4 addresses they need to serve their customers - basically they can abstract hundreds to thousands of customers behind one single public IP address rather

than giving each one their own address. But in the end, this architecture didn't allow for port forwarding to the public IP address, leaving me with this problem. See this graphic I found on the internet that shows a little bit of the difference in topology (thanks to netelastic.com):



The main difference is that my old ISP has plenty of IPv4 addresses (to the tune of 1.5 million) for each of its customers, while my new one has a modest set (~8,000), so CGNAT ensures they will not run out of public addresses.

Part 2: Solving Public Access (Websites)

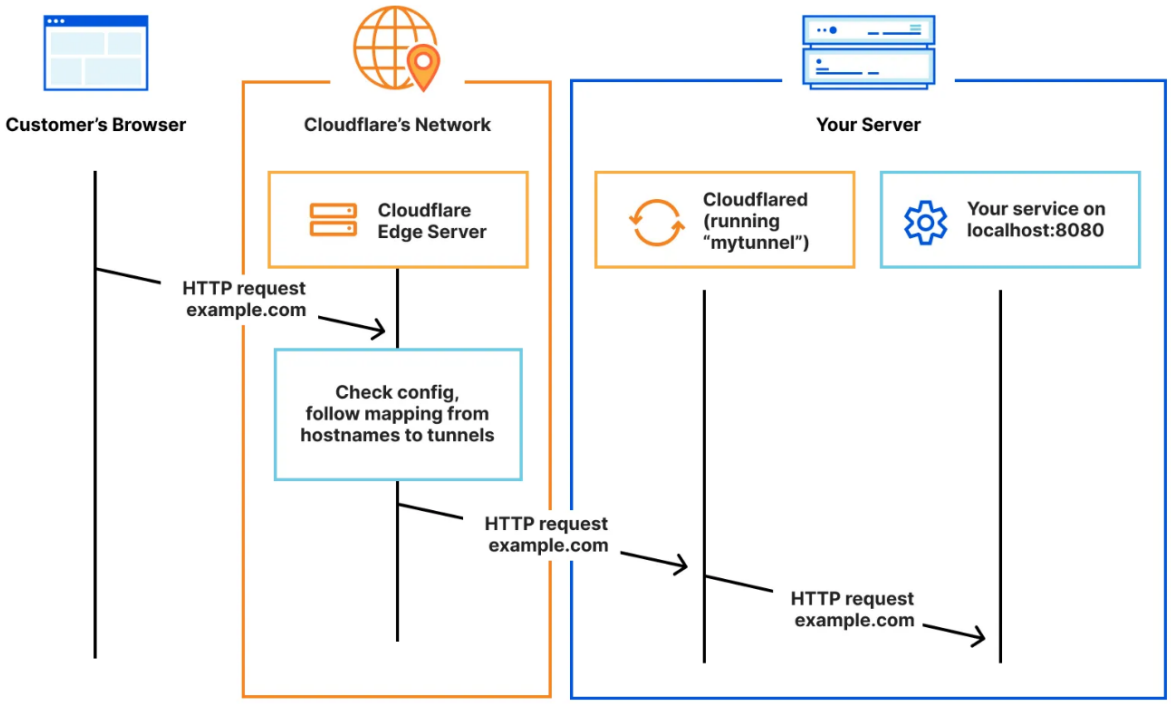
With this new problem, I was no longer able to just forward ports 80/443 to my home router, and set up DNS entries to point some subdomains to it's public address. With this, we had to navigate a few different options in bringing these sites to the public internet – after a few hours of research, it realistically came down to these options:

- **Buying a static IP address** – would be the easiest in terms of just lifting and shifting my current set up, but costly. I would assume this would take a business-level internet plan to accomplish.
- **Spin up cloud-based reverse proxy** – I could spin up a VM in Azure that pulls the websites from within my network and then serves them on the edge to internet users. Again, this would be costly, but also would add a layer of complexity that could making scaling very hard to do in the long run.
- **Cloudflare Tunnel** – In a nutshell, this is basically a reverse proxy as a service in a way (RPaaS? who knows…) that abstracted the management of an actual server. This was free up to a certain extent – and we were already using them for DNS.

If you couldn't tell by now, the answer was the last option – Cloudflare Tunnel. Essentially, I point my subdomains to a Cloudflare edge server, then the edge server has the ability to send HTTP requests to my internal web server, which in turn returns the site to the edge server, then to a user's browser. Not only does this restore access to my websites from the public internet, it also introduces a few really nice security advantages when it comes to self-hosting a web server on my own:

- No more open inbound ports to my network, reducing possible attack surface
- DDoS protection
- Encrypted traffic between my web server and the edge
- The possibility to integrate further Zero-Trust features.

This is how Tunnel looks in practice (thanks to Cloudflare documentation):



As you can see, a Cloudflare agent lives on my web server, and it's purpose is to facilitate *encrypted* communication between my Nginx service and the Cloudflare edge server. In Cloudflare's dashboard, I did have to configure what sites go where in terms of ports, and other miscellaneous configurations, but this ultimately fixed the issue of not being able to serve websites from our own web server. While the websites we hosted were now visible to end users, we still didn't have a way to remotely manage the environment if needed since our traditional VPN service was now rendered inoperable.

Part 3: Solving Private Access (VPN)

Now that we solved the problem of not being able to host websites, we still wanted a way to access the environment remotely in case there were any issues. Previously, I was able to implement Wireguard – which, again, was fairly straight forward in the old ISP set up. I forwarded the needed port all the way to my home router, and set up a subdomain to point to my home's public IP address. I would still have to notate the port on the VPN client to get to the Wireguard service (i.e. vpn.blakeniebrugge.com:51820), but it worked for me.

Once we got the new ISP, this once again broke. We couldn't Tunnel the VPN through Cloudflare like the websites mentioned in the last part since most VPNs utilize UDP by design. Cloudflare tunnels have to maintain a connection, making this impossible. I was back to the drawing board.

I had done a little research, gave up, ended up doing more research, and found a service named Tailscale that could possibly help. It is built on Wireguard's VPN capability, but it's calling card is that it doesn't need a public-facing server to connect to in the traditional VPN sense. It did this by introducing "overlay networking" which means it was brokering connections between nodes, regardless of where they were, how they were set up in their network, if they were behind firewalls, proxies, CGNATs, etc. Each node in a Tailscale tenant has an agent, and when you wanted to connect the two, they reached out to a Tailscale "coordination" server which passed on info to each node on how to connect to the other. This is how it was set up in our case:

I had installed the Tailscale app on my phone, and an agent on my VPN VM living in the Proxmox VE. We now had two devices that could connect to each other, but nothing like a VPN – it was only communication between these two. To explain how they all the sudden are able to connect, it goes like this:

- VPN server connects to the Tailscale coordination server, and it give it the return address to itself (i.e. <New ISP's Public IP address>:<port number> or something like 206.125.156.1:12345)
- iPhone also connects to the coordination server, and gives it's return address, much like the server
- The devices then really just use this information to set up a Wireguard connection between each other.

If direct connection isn't possible via the steps above, the devices would then connect to a Tailscale DERP (Designated Encrypted Relay for Packets) Server that acted as a middleman, forwarding packets between the two hosts. Luckily, I was able to manually see this direct connection from my VPN:

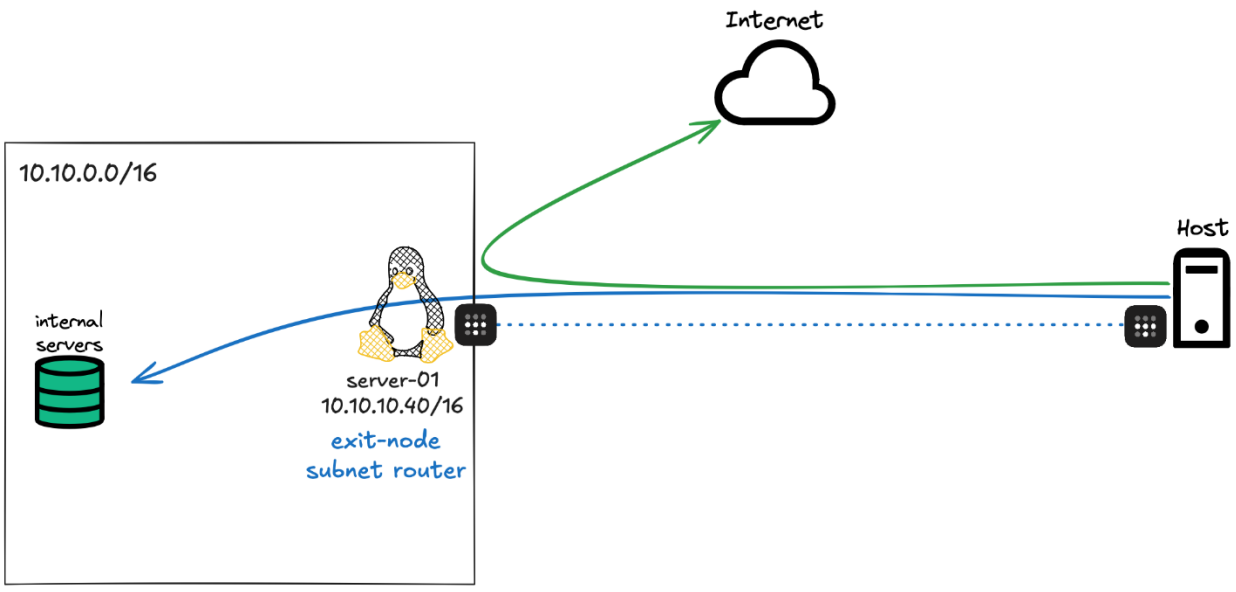
This was the biggest piece of the puzzle. Once we could connect the two, it was just a matter of reconfiguring the server side. I configured it as the exit node, which meant that all internet traffic would leave out the VPN server, onto my home network, and out to the internet. We now had a very basic VPN-like connection. One last thing I set up was having

the VPN server advertising routes to our Tailscale network which was telling all other nodes (just my iPhone for now) that it could reach 10.0.0.0/8 through itself. Now from my iPhone I could reach my internal VMs once again – all without having to implement anything too complex and also adding security + scalability yet again.

This is what the set up would have looked very similar to, with some clarifications:

- Server-01 would be our VPN server in our case
- Host would be my iPhone (but could be added to down the road)
- I advertised the entire 10.0.0.0/8 block from server-01, not just 10.10.0.0/16

Credits to Suresh Vina in his article "Tailscale VPN: A Network Engineer's Perspective":



Conclusion/Reflection

In the end, we had successfully restored functionality to our self-hosted websites and our VPN service. With this, we also added extra security layers by not exposing ports directly to the internet, leveraged encrypted tunnels, and even abstracted the control plane of the VPN service out of the equation. Another positive is that we are now way more scalable – no more having to reconfigure firewall/routing rules when we need to add devices or websites. Most importantly, we learned a lot about different topics and methodologies that I hadn't even heard of before. This includes:

- Traditional VPN vs overlay/mesh networking
- NAT traversal and CGNATs
- Reverse proxying leveraging Cloudflare Tunnel as opposed to traditional port forwarding

This also shows ability to adapt and problem-solve in evolving environments, familiarity with modern/emerging networking and security approaches, and also the ability to not only design/theorize secure and cloud-inclusive architecture, but to actually implement it.

If there are further comments, feel free to reach out using the contact information found below the abstract on page 2. Looking forward to speaking!