1 Overview

This is the description of how I created my k3s cluster with tailscale and old thinkpads. I believe that the architecture is somewhat unique, and fits the use case of broke college kid trying to deal with no public ips and moving all the time. These docs are currently a work in progress, and contain a lot of my musings on the trials and tribulations taken when building this cluster.

2 Hardware

2.1 Compute

One of the biggest priorities of this project was to reduce the cost of physical hardware. The cheapest way to do this was to buy used thinkpads (to which I am preferential anyway). Two of the secondary nodes are used thinkpad T410s and the master node is a cheap fanless computer. I decided to go with a new fanless computer for the master node to increase reliability a bit more. The thinkpads fail often, and I want to increase reliability by one metric.

These thinkpads have been semi-reliable for the last year during the formation of this cluster. I highly recommend them in your projects. You can also look into removing the Intel ME as well as installing coreboot if you have a computer that is old enough. You can also go as far as removing the display with little issues (I did not do this however, as I prefer to keep the display for debugging).

2.2 Storage

I removed all the hdds that the nodes had as I understandably did not want to deal with spinning disks. In it's place, I have installed ssds for both the boot and storage drive of all the nodes. I do this more for reliability than speed.

2.3 Physical Networking

I actually use a Access Point that has a dual purpose as a switch and an Access Point in order to connect the nodes of my cluster. I am not obsessed with networking speeds (because my computers are not that fast anyway), so

I don't worry about max bandwidth too much. In practice, my setup really looks like a bunch of laptops sitting on top of each other, underneath my nightstand.

3 Networking

3.1 Tailscale

3.1.1 Reasoning for Tailscale

This is the barebones of the project. One of the biggest restrictions that I had when creating this cluster is that I can't use port forwarding in apartment currently. This meant that I needed another way to expose my cluster to the public internet. There are a ton of ways to do this. One such way is to use ngok or something similar. However, I realized quite quickly that Tailscale could also afford me some privacy measures in addition to allowing me to expose traffic to the public internet.

3.1.2 How the hardware is connected

All nodes in the cluster and machines that are used for administration in the cluster are connected in a tailscale instance. They communicate through tailscale magic dns and can be located anywhere connected to the public internet. In addition to this, any of the nodes connected with a public internet address can be used as an "entrance node" (in this case I use an aws ec2 instance for stability).

In addition, I run a headscale instance on that same ec2 instance in order to do the actual key distribution (and the services that you would usually have to pay for on tailscale). This allows me to connect more than 5 devices, and keep my cluster expandable. I also like it, because it means I can self host (and I prefer that). In addition, I use the Tailscale DERP servers, because they were more reliable than what I was able to deploy on headscale.

3.1.3 How data moves and TLS is handled

The entrance nodes forward all of the tcp packets that they get on port 443 (https) to the cluster in order to do ssl termination there instead. I also use proxy_protocol in order to keep domain information. I prefer to do ssl

termination on the cluster itself so I can take advantage of things like cert-manager as well as being able to avoid wildcard certificates. In addition, I can create some separation and therefore security, because I only forward the ports and allow the domains I choose. Although limited, it means that I might have a little more protection than a simple port forwarding solution. I don't have to expose ssh access to the entire world. In addition, I don't have to expose cluster access to the entire world as well.

3.1.4 Internal Tooling

Lastly, tailscale also means that there is a very straightforward way to give people access to internal tooling. I can make only some domains available to public access (by whitelisting them in the entrance node), and then have all other domains only available as internal tooling.

3.1.5 Tips and tricks

As a couple of tips, it can be hard to keep the nginx config consistent amongst all of your entrance nodes, so what I recommend doing is keeping git as a source of truth, and do a cronjob that pulls from a git repo and automatically updates at whatever frequency that you would like. I am not sure if there is a better way to do this (maybe create some sort of webhook solution), but this is what I ended up with. In addition, I would avoid keeping your nodes in restrictive networks, as this means that they use the Relay servers as little as possible and you have faster speeds.

3.2 Klipper

This is somewhat standard, but I use klipper (or servicelb) as the bare metal load balancer for this project. This sits on the tailscale magicDNS and works perfectly to distribute traffic accross the nodes. I also have multiple targets in the nginx config so that traffic is somewhat balanced from there as well. If one machine goes down, theoretically traffic should be forwarded to the node that is up and everything should proceed as normal (this has happened once - and to my surprise everything worked as expected).