Cheap Portable k3s

Sachin Iyer

July 20, 2023

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 cheap 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.

1.1 Cheap

I needed this cluster to be somewhat cheap for two reasons. First, I don't have any super high SLA requirements, nor do I really care too much about performance. As such, it is better for me to try and create something that optimizes for price.

1.2 Portable

I tend to move around quite a bit, and it is an absolute pain to set up port forwarding everywhere that I go. I also am not guaranteed access to a public IP wherever I am going. Therefore it is necessary that I am able to move my cluster with ease.

1.3 Secure

The previous version of this cluster used to be accessible by ssh with a 5 letter alpha password. This is just not smart. I wanted to make this cluster slightly more secure and control the traffic that hits the cluster better.

1.4 Based in Open Source

I wanted to try to use open source software for pretty much everything (down to the bios). This was because I wanted to not only support open source projects, but see if I could really run this without having to rely on any proprietary software.

1.5 Resiliency

Ideally I wanted to design something that had non-resilient cheap hardware and a very resilient architecture. I wanted to make sure that the cluster would have a much higher uptime than would be expected of it.

1.6 k3s

I really wanted to make a kubernetes cluster not just to learn, but because it allows me to deploy all of my apps with ease. I also get to keep my data on my machines and overall just have a testing ground for all of my stuff that I do. If I want something to go up, I am not at the mercy of a cloud service.

2 Applications

These are the applications that are currently deployed.
There are configs in another repo.

1. My website - This is my personal website - https://sachiniyer.com
2. Random Projects (e.g. control-display or my toxic tweets project) - I can deploy my random projects with ease now
3. Hugo blog - This is my blog that I write posts (mostly to myself) on - https://blog.sachiniyer.com
4. Emptypad - a scratchpad that I can use on the web (built by @aminoa) https://emptypad.sachiniyer.com
5. Privatebin - Basically PasteBin, but I want to control my data, and I don't want it to be indexable. https://bin.sachiniyer.com

6. Kutt - Basically bitly, but, again, I want to control my data, and I don't want another company taking it. https://s.sachiniyer.com
7. Nextcloud - This is basically my self hosted google drive - https://store.sachiniyer.com
8. Gitea - My git hosting solution that looks pretty - https://git.sachiniyer.com
9. VaultWarden (private) – With a hardware key is how I store all my passwords. https://pass.sachiniyer.com
10. nfty (private) – Basically a way to automate notifications to my phone for downtime and anything else I need to remind myself about https://nfty.sachiniyer.com
11. prometheus (private) – How I get metrics about the cluster and anything else. https://metrics.sachiniyer.com
12. Jupyterhub (private) - For all the ML stuff I am trying to learn - https://hub.sachiniyer.com
13. Dav Server (private) - How I sync my tasks and calendar, and files. - https://dav.sachiniyer.com
14. Minecraft Server (private) - Minecraft server, because my friends wanted to play (I'm really bad) - https://sachiniyer.com:25565

3 Hardware

There are two thinkpads and one fanless computer in my 3 node k3s cluster. I also have an ec2 instance that acts as my "entrance node" and also headscale control server. I don't include this in the cluster, because I want an odd number of nodes.

3.1 Compute

I have a very powerful 12 virtualized cores in this cluster. It may not seem like much, but I paid $250 in total.

Thinkpads The cheapest way to get cores is to buy used thinkpads (to which I am preferential anyway). The two secondary nodes are a thinkpad T410 and a thinkpad T420. These thinkpads have been semi-reliable for the last year during the formation of this cluster. You can also look into removing the Intel ME as well as installing coreboot if you have a computer that is old enough (the T410s are). The computer will also boot if you short two of the pins in the eDP ribbon connector (which means you can remove the display).

Fanless Computer from China 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. It is also because there were no T410s on new york craigslist when I was expanding to my last node. **Don't buy the AWOW Mini PC - AK41. It is super unreliable and sucks**.

Why no raspberry pi I did not use raspberry pis because a lot of my applications do not play nice with arm, and they are extremely expensive right now. In the previous iteration of this cluster I ran a multiarch setup with x86 and arm, and I ended up running into a lot of weird problems. I decided to avoid the headache and just stick to x86.

3.2 Storage

I don't want to deal with hdds - I removed all of them. Spinning disks are really quite annoying. Each node has an internal boot drive (128 or 256 gb) and then an external 256gb ssd. In total I have 768gb of storage available.

3.3 Physical Networking

TP Link Access Point I use a TP Link Router (in AP mode) to both extend my wifi network in my apartment and also connect the nodes. I don't worry too much about networking speeds, but everything is theoretically around gig-speed.

TP Link Switch I also have a small switch that I use for the rest of the machines and some raspberry pis that I have.

4 Networking

This is the heart of this project. The main motivation for remaking this cluster was to create something that I could move from place to place with me very easily.

4.1 Tailscale

4.1.1 Reasoning for Tailscale

What is Tailscale Wireguard is a way to establish quick p2p encrypted tunnels at a kernel level. Tailscale handles the key orchestration for wireguard. The end result of this is a p2p connection with every other machine in the "Tailnet". Using some fancy port opening stuff, you can also expand your wireguard tunnel through NATs and essentially communicate with every other machine no matter where you are. Each machine gets a Tailscale IP as well as a MagicDNS (an entry in resolv.conf basically).

Why Tailscale When all the nodes of the cluster and machines used for controlling the cluster are in a "Tailnet" together, you essentially have a private network where not only can you access internal service (one of the advertised uses for tailscale), but also machines can talk to each other without worrying about where they are. You can also manage individual nodes without needing to be in the same network. There is no more tricky, finicky, insecure port forwarding.

What's Headscale Headscale is an awesome open source project that allows you to self deploy a server with many of the same capabilities as the proprietary Tailscale server. This means that I don't have to worry about a device limit and can keep all keys on my own machines. Since the Tailscale clients are open source, I actually can just use the Tailscale clients to connect to my headscale instance (including the android app).

Another cool thing about Tailscale is that I can still use their DERP and relay servers without issues. Although headscale can be deployed with a DERP server, I found this to be very finicky.

4.1.2 How do packets move

Entrance Nodes This is the term that I started using for traffic that comes into the cluster. These nodes need a public IP and basically do an nginx proxy_pass. I use an ec2 instance for this purpose. I may swich back to a machine with port-forwarding if I find a place that I can get a public IP with.

Another way to do this would be to just configure IP tables to pass the packets from the public IP to the tailscale IP. I don't do this however, because I want to validate it as an HTTP request first. I also prefer to configure nginx and route based on domain name instead of blindly forwarding packets.

You can actually visit a couple of non-cluster machines that are connected to the network through this entrance node, such as my laptop (if it is up), Raspberry Pi 1, and Raspberry Pi 2.

Exit Nodes This is the tailscale term for machines that advertise themselves as being able to forward your traffic through them. I actually use a rapsberry pi for this purpose. ~~I also connect that raspberry pi to my preferred VPN, so that anytime I want a VPN connection, I don't have to use an external application. I can also add a VPN connection very quickly to any node just by configuring it to use the exit node.~~ I put VPNs on the cluster itself so it is a lot easier to choose which ones I want. I also am considering putting exit nodes directly on the cluster.

TLS Handling TLS is a bit tricky. I prefer to do ssl termination on the cluster or end machine instead of the entrance node itself, This way, I can keep the traffic encrypted while traveling to machine as well as take advantage of cool tools like cert-manager. On nginx, you use a combination of proxy_pass and SNI (Server Name Indication) to get your packets going to the right place. You can also configure Traefik (my ingress of choice) and Nginx (my other ingress of choice) to accept proxy_passed packets. Also depending on the Load Balancer that you are using, you may have to configure that to accept proxy_passed packets as well.

Domain Names Another requirement for this project was to be able to seperate internal and external tools. I do this through whitelisting domains at the entrance nodes. Instead of blindly forwarding all the traffic from *.sachiniyer.com, I instead just forward the tools I want available to the

world. Traffic that reaches the entrance nodes with those domains will be allowed into the tailnet.

All traffic with other domains will have to be generated from inside the tailnet. You can make this easier with a quick change to resolv.conf to point traffic from your domain to the magicDNS of the cluster. This results in a clear separation of internal and external tools. You could also configure your dns record to only have the whitelisted domains as well.

4.1.3 Tips and tricks

As a couple of tips, it can be hard to keep the nginx config consistent among 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. There may be a better way to do this with webhooks. I would also 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.

4.2 ~~Klipper (or ServiceLB)~~ MetalLB

Load balancing is very cool. All load balancing communication happens through Tailscale MagicDNS. ~~I used to use MetalLB instead of the k3s default of klipper. However, Metallb would often overwrite my proxy_protocol packets, not allowing me to do TLS termination on the cluster. I was actually able to get Klipper working with a lot of ease.~~ I figured out the proxy protocol issues and am back on MetaLB. I am considering a switch to cilium because it has a much more interesting design pattern (eBPF-based load balancing).

4.3 Traefik Ingress

I made a similar switch from Nginx to Traefik for the same proxy_protocol reasoning. the Nginx ingress was not respecting my proxy_protocol packets, and somehow the Traefik ingress was really easy to configure and get working. I also started to like Traefik a lot more because it seems to play better with the more modern versions of kubernetes ($\geq$ 1.25).

4.4 Cert Manager

In the previous iteration of this cluster, I fell in love with cert-manager. I desperately wanted to use it again. This was one of the main reasons for doing TLS termination on the cluster instead of the entrance nodes (and also that wildcard certificates are a bit insecure). I integrated cert-manager with Traefik and it works quite well. I love being able to spin up a cert easily and have ssl easily enabled.

5 Storage

I don't do anything to crazy here. The most important thing is being able to do good backups to the cluster.

5.1 Rook Ceph

I read quite a bit on ceph when I was interesting in storage managers, and quickly fell in love. I also really love rook-toolbox and overall I think that rook is just an excellent storage manager.

5.2 Why not longhorn

I didn't choose the rancher suggested longhorn, because ceph is a bit more resilient. I also want in the future to get more into the weeds with my storage manager configuration and longhorn does not allow me to do that effectively.

6 Future Goals

6.1 move master node

The machine that the master node is on right now became unstable, so I need to figure out a way to migrate that over to something more stable.

6.2 more reliable Headscale

Currently Headscale is deployed on bare metal which is okayish. However, I would much rather it be deployed in a docker container, or spun up with docker-compose or something with a bit more fault tolerance.

6.3 ec2 autoscaling

I want to integrate autoscaling with ec2 instances for when I am really out of compute on the cluster. This is going to take a lot of work however, because I will need to figure out a way to automate the addition of a node to the Tailnet, as well as figure out how to automate the addition of a node the k3s cluster. These are both very possible, but a little bit hard to do super securely.

6.4 Move entrance node

I want to move the entrance node off of the ec2 instance into another node that has a public IP. I think that this is a little better cost wise, and means I am fully not dependent on AWS. I will need a public IP for this to work however.

6.5 Add two more nodes

I quickly reach the maximum limits of this cluster, and I think that it would be prudent to start planning for the addition of two more nodes

6.6 Better Storage

I think it will be nice to add some better storage to this system. The current storage capabilities are alright, but it would be nice to do intermittent backups to the cloud as well as potentially add even more capacity. SSDs do not cost much these days.

6.7 Integrate Wireguard Key Management into k3s

~~Lastly, the by far most ambitious improvement would be to move the headscale control server functionality into k3s and handle it natively. This would require a lot more work and system design, but I think it could be interesting to actually make wireguard tunnels through k3s native system design. I will have to think a lot more about this though.~~ - *UPDATE* seems to be done by someone already.