My Homelab Journey

My Homelab journey and path I took building it.

- First Try in Homelab Journey
- Foundation
- First Virtual Machine and Applications
- Search Engine & Public Access
- Access Control & Identity Management
- Application Sprawl
- Docker Management
- Backups & Disaster Recovery

First Try in Homelab Journey

The idea for my homelab began when I first saw Pi-hole at my friend's house and discovered how it works. He had set it up on an old laptop as a simple DNS server designed to remove ads. I was amazed by the data it produced and the ability to curate my own lists of malicious sites for DNS filtering. More importantly, it sparked a curiosity within me, revealing just how little I knew about that aspect of technology.

At the time, I was working as a tech support specialist at a small college, primarily engaged in troubleshooting, password resets, and hardware repairs. I had a genuine interest in the hardware side of computers. I enjoyed taking them apart, putting them back together, fixing small components, and exploring the intricate parts of motherboards. I already owned a custom-built PC that mainly served as a gaming machine, but it also doubled as a personal project, allowing me to learn about the various components and how they fit together.

Despite my enthusiasm for hardware, I felt like a stranger to system administration, networking, and the server side of IT. While I had a basic understanding, my knowledge was limited. I can admit that I was somewhat intimidated by networking, something that resembled an intricate spider web. During this time, I began focusing more on cybersecurity and data integrity, motivated by my experiences at work as well as my personal concerns. After losing access to some personal files, I became reactive and realized I needed to learn how to be more proactive. I explored numerous articles and online videos, which only deepened my curiosity. This led me to my first introduction to DNS, its purpose, and a desire to delve into something new. Acknowledging my ignorance, I understood that the only way forward was through knowledge. While I believe mistakes can be valuable learning experiences, I disagreed with the notion that I had to make them to progress. I spent time researching online, learning how to set up my server and run a Pi-hole DNS server. I watched YouTube tutorials demonstrating various methods and gained firsthand experience from my friend’s setup.

In my second job, which was similar to my first, I was assigned an exciting project: decommissioning an old data center and setting up a small data center for a few professors to use in their labs. This opportunity allowed me to set up servers and create a small cluster for myself. Although it was isolated from the internet, making it perfect for testing, I found it to be my first real experience with a homelab and server management. While this was technically a school environment, my work on this project is best described as a homelab since I was running several open-source applications on my own.

I installed Proxmox on the system as the main hypervisor, setting up a cluster of two servers and one running Proxmox Backup Server. Although I found building a server rack and assembling hot-swappable drives fascinating, I was equally intrigued by how the software side of servers operated. After configuring the hypervisor, I installed Pi-hole on a virtual machine and began using it. Since this system lacked an internet connection, I tinkered with an old laptop I had in the server room.

Although this setup was temporary and lasted only a few months, it provided me with invaluable experience in networking, virtualization, and server management. During this period, I was also enrolled in a graduate program in Information Management, which complemented my hands-on experience with relevant classroom material.

Foundation

The first go-around taught me that there is much more involved in establishing a homelab than I initially realized. While the essential requirement is a device capable of running a server, other critical aspects include networking, security, and constructing a supportive infrastructure. By taking the CompTIA Security+ exam and enrolling in several cybersecurity courses during graduate school, I came to understand that a core part of security lies in the correct setup and backbone structure. To develop a strong and secure homelab, I had to start from the foundation. Much like the OSI model, I needed to begin at layer one and build upward.

Choice of Hardware

Choosing the right device can be challenging, but I was pleasantly surprised to discover that even an old, simple PC can be transformed into a fully functioning server. While I couldn't control physical malfunctions, I concentrated on what I could manage. One of my first steps was ensuring there was a backup server running, ready to replace my primary server if needed. It's quite astonishing how effective some used mini PCs can be as powerful servers. They offer low power consumption, compatibility, and ease of upgrades. Buying two mini PCs on eBay with robust specifications and a healthy amount of memory was perfect for starting out. These two PCs are still operational years later. Initially, I didn't focus much on specifications, but it quickly became apparent that memory is consumed far more rapidly than storage.

The best place to start is with your old laptop or PC. It's quite interesting how easily an old laptop can be repurposed. I had a Mac Mini, which Apple discontinued a few years back, making this device no longer usable, at least not with macOS. I attempted to use FlashOS to install the latest macOS on a device that was not supported, and while that worked, there were many ups and downs along the way. However, installing any Linux OS and repurposing an outdated device that had reached end of life, surprisingly, was easier than I thought. Thirteen years later, that Mac Mini runs smoothly and without issues. Additionally, I removed its Wi-Fi module, which provided me with extra space to install a second storage disk inside.

I found a lot of posts online, and many people suggested running a Raspberry Pi. While these devices are efficient, repurposing the device I already had was a much cheaper option and also allowed me to expand my hardware components. This is another significant reason to consider pre-

2020 mini PCs. Newer devices are great, as they are more compact and power-efficient; however, with each new model, more components are being soldered to the motherboard and cannot be replaced. This is especially true for new DDR5 RAM sticks. From my experience, RAM memory is the first thing I run out of and need to upgrade. While acquiring another device is an option, upgrading my current PC from 8GB to 16GB or even 32GB is a much easier and more efficient process.

Networking and Logical Security

The second part of building a solid foundation focused on the next two layers of OSM model: networking. The best way to overcome your fears is by confronting them. I began by educating myself through reading and watching videos, and then I purchased network equipment that allowed for full configuration capabilities. Through my research, I discovered that Ubiquiti equipment was the best choice for my journey. They offer enterprise-level networking hardware and software tailored for home use, all at reasonable prices and with a user-friendly interface. Unlike Cisco, the industry leader, using Ubiquiti didn’t make me feel like I needed years of experience to operate it. A friend also helped with my decision by offering me his UniFi router, requiring only a switch and an access point for complete functionality. Utilizing the Ubiquiti system enabled me to secure every aspect of my network. From configuring VLANs, setting up layer 2 and layer 3 security features, to controlling who connects to my network, I was provided with an enormous amount of data and insight.

While the DNS component wasn’t as comprehensive as Pi-hole, it proved stable and effectively limited application usage. This setup also presented an opportunity for me to learn more about firewalls—specifically how to establish rules, determine which ones to implement, and maintain necessary traffic flow without compromising security beyond standard practices. Early in my network setup, Ubiquiti upgraded their firewall to a zone-based firewall, which opened up a whole new Pandora’s box of options.

The final aspect of building a strong foundation for my homelab involved a proper application setup. A well-configured Proxmox environment, complete with a backup server, seamless migration capabilities, and a firewall installed on Proxmox, was essential. I ensured that my hypervisor management UI was physically and logically separated from the rest of the virtual machines (VMs) on the network. This configuration allowed me to restrict access to the Proxmox management UI to only a few devices on one VLAN, while the VMs running on the server were assigned to their dedicated VLANs. This segregation was particularly vital given my access to the fully configurable zone-based firewall, enabling me to protect specific VMs and provide them with the appropriate level of security while ensuring adequate accessibility for externally-accessed applications, all without compromising the functionality of other apps.

Before I began setting up my servers and network, I was quite a stranger to networking and timid about diving in. However, after completing my setup, I realized how enjoyable networking can be and how eager I was to pursue a career as a network engineer. My journey not only revealed how much I didn’t know but also ignited my passion for network configuration. I learned that a significant portion of security comes from proper network architecture and foundational structure.

Hypervisor Setup

I already had experience with Proxmox software, and their community is fantastic, always quick to answer questions on forums. Additionally, many YouTubers I follow use Proxmox and have produced great setup videos. One channel, in particular, Learn Linux TV, was my main guide in setting up my Proxmox server and ensuring its proper security. The installation of the software is pretty straightforward; once you create a bootable flash drive, you're ready to start. I made sure that this device was on a static IP and outside the DHCP range, which facilitated an easy installation and avoided any future reconfiguration if the IP changed. While this can be configured from the Proxmox installation side, I also set it up in Ubiquiti to ensure that my switch wouldn’t accidentally assign another IP.

Once I logged into Proxmox, the first step I took was to create two additional admin users: one PAM user and one PVE Realm user. The generic account is root, which resides in the PAM realm. I created a secondary PAM user through the command line, granting it admin access in the Proxmox application as well as sudo access through the terminal. After ensuring the setup was correct and that the user worked as intended, I disabled my root account and configured SSH access to only use public key authentication (PK). While the PAM account is sufficient for most operations, I primarily use it for SSH and terminal access. I created a PVE Realm user with admin access for web UI activities. The only limitation of this user is the inability to perform updates, but I can easily SSH into the device to carry those out.

Once my users were configured, I set a few firewall rules in Proxmox at the database level. My machine was already secured with network firewall configurations, but as a good practice, I added additional firewall rules to enhance security. I ensured that only one device could access Proxmox via IP and SSH. Additionally, I made sure that any other service or access point went through a domain with a valid SSL certificate, keeping all communication secure and encrypted. Is this level of security strictly necessary? Not really, as Proxmox uses its own self-signed certificate and protects against threats using HTTPS. However, adding a Let’s Encrypt certificate with my personal domain provides additional security and eliminates that "Potential Risk" message that appears in browsers. All this was achieved without any port forwarding, which I’ll discuss in a later chapter.

The last thing I did was replicate the same configuration steps for the Proxmox Backup Server. When I configured it, it was still in its early stages but was already in a stable version and performed well. Much like the Proxmox Virtual Environment, the Backup Server installation was straightforward. After completing all the same security steps I implemented on the PVE, I connected my server to the Backup Server and scheduled daily backups of every machine created across all clusters. Although I only had one machine initially, this setup allowed me to set it and forget it, making it work automatically—even for future use when I acquire another device and create a cluster. While backups are configured on the Virtual Environment side, prune jobs, verification, and other backup checks and optimizations were set up in the Backup Server.

Once everything was set up and running, it was time to create my first virtual machine and start my homelab applications.

First Virtual Machine and Applications

Once the network, hypervisor, and security were configured, I began spinning up my first virtual machine (VM). I chose Ubuntu Server as the base image because of its stability and reliable performance. Later on, I experimented with a few Fedora Server VMs, but I ultimately found Ubuntu easier to work with.

Before installing any applications, I created three VMs and assigned each a specific role:

- **Overseer** – This VM hosts critical applications for administration, monitoring, and notifications across my entire homelab.
- **Internal Server** – This VM is dedicated to applications meant solely for my personal use. It is not exposed to the internet and does not allow guest access.
- **External Server** – This VM runs applications intended for open or guest access and is securely exposed to the internet without relying on traditional port forwarding.

Thanks to my zone-based firewall and preconfigured VLANs, each machine was assigned its own VLAN. As my homelab grew, I deployed additional VMs that followed one of these three core functional roles. The idea was that Overseer VMs (VLAN 710) could access machines on both the Internal (VLAN 720) and External (VLAN 730) networks, with only limited return traffic allowed.

Reverse Proxy

Once the servers were installed and configured, I began setting up a few core applications that I considered essential for maintaining the homelab. While most of my applications are accessed internally—and the externally facing ones run through a Cloudflare Tunnel—I still wanted each service to have a proper domain and a valid SSL/TLS certificate. Even if an app already had its own certificate, using a unified setup made everything more secure and eliminated browser warnings about potentially unsafe pages. It also made the whole environment feel more polished.

The first application I deployed was **NGINX Proxy Manager**. I installed it using Docker, which at the time felt like the simplest approach, though not as convenient to manage as Docker Compose—which I adopted a bit later.

NGINX Proxy Manager came with a small learning curve, but once I figured out how to configure DNS-01 challenges through Cloudflare and use the proxy internally, assigning hosts to their subdomains became very straightforward.

Originally, I used Pi-hole running on a VM in the Overseer VLAN. However, Ubiquiti later released an update that added configurable DNS records, including CNAME support. This made DNS management much easier, so I eventually switched from Pi-hole to the UniFi DNS server.

Docker Manager

Although I'm comfortable using the command-line interface (CLI), I prefer having a user interface (UI) for day-to-day management. That's why I installed Portainer. It made deploying images and maintaining them through Docker Compose—called Stacks within the app—much easier.

I connected all three servers as environments to the main Portainer instance running on the Overseer VM using Portainer Agents. Setup was simple, thanks to the abundance of guides available online. Once Portainer was in place, I moved on to the next important application: Uptime Kuma.

Notifications

While nothing in my homelab is truly mission-critical, I still wanted to know when an application became unavailable or stopped working. I deployed Uptime Kuma through a Portainer Stack and configured it to send notifications to a dedicated Discord channel whenever a server or application went down or came back online.

Kuma is lightweight, easy to configure, and supports many notification integrations. Discord was the simplest option for me at the time, especially since I didn't yet have my own mail server.

Search Engine & Public Access

Search Engine

Around this point in my homelab journey, I had the baseline established and was ready to focus on applications that would bring more meaning to my homelab and actually be of use. Throughout graduate school, a big focus of mine was cybersecurity and privacy, as well as data analytics. Once I gained a better understanding of what data we leave behind every time we search something on Google, it became clear to me that I needed something more private and within my control. While I mostly use Brave browser and DuckDuckGo, I still wanted something I could fully own. That led me to SearXNG.

SearXNG is a free and open-source meta search engine that I discovered through watching homelab videos on YouTube, specifically from NetworkChuck. After watching his video, the setup seemed pretty straightforward — install Docker, run a simple bash command, and you have a containerized application up and running. I already had everything in place: a domain connected through my reverse proxy, notifications set up to monitor the container in case it lost connection or went down, and a virtual machine that was already provisioned and ready to go.

Since this is something I wanted to access remotely and potentially share with friends, I deployed it on my External VM, which was already in the proper VLAN with the appropriate permissions and security controls in place.

While it all looked simple at first, there were a few setbacks. I had to dig through Docker logs to figure out what errors were occurring and how to fix them. One valuable lesson I learned is that every environment is different, just because something works for someone else doesn't mean it will work for you without adjustment. After fixing errors and using ChatGPT for help in understanding the issues, and reading through hundreds of lines of logs and configuration, I finally got it working. SearXNG was live.

It's a simple search engine with no data collection or trackers. The application works by fetching results from multiple pre-defined search engines, creating an anonymous one-time session profile, and returning those results without leaving any cookie crumbs behind — no record of who searched, from which IP address, or anything else that could be traced back to you.

Tunnels

While the traditional way of exposing applications to the internet is through port forwarding, it isn't the safest approach. With security and privacy at the forefront of my homelab philosophy, I wanted

to do this safely and securely. To achieve that, I went with a more modern approach: application tunneling.

My domain was already registered with Cloudflare, and their free tier offers a lot of useful features — one of which is Cloudflare Tunnel. This allowed me to connect a forward proxy service running on Cloudflare's infrastructure back to my personal network, exposing my self-hosted application to the internet without any port forwarding required. Nothing on my router needed to be opened, which significantly reduces the attack surface.

To make this work, I had to install a lightweight agent cloudflared, running in its own container. This agent communicates outbound to Cloudflare to establish a secure tunnel and expose whichever application I've configured. Once everything was in place, I followed the steps in the Cloudflare dashboard and set up the appropriate connection. Cloudflare automatically created a CNAME DNS record, which allowed anyone to access my search engine through a public URL. Even though SearXNG was running on a private IP (e.g., 10.0.0.x), requests from the outside world resolve to Cloudflare's public infrastructure, making it much harder for anyone to discover or target my actual server.

On the Cloudflare side, I took every available step to lock things down as tightly as possible. I created access rules for my tunnel applications and restricted incoming traffic to only a handful of countries, specifically, the ones I'd realistically expect legitimate traffic to originate from. This makes the service more resilient to automated attacks and probing. Cloudflare also provides native DDoS protection, keeping the application safe from botnet floods without any additional configuration on my end.

Docker Management

This project was fairly straightforward to get running, but it highlighted several gaps in my initial foundational setup. While Docker containers can be managed entirely through the command line, deploying and maintaining them through a GUI is much more practical for day-to-day operations. This led me to install Portainer.

I set up the main Portainer instance on my Overseer VM, with lightweight agents deployed on both the Internal and External VMs. This gave me a single pane of glass to manage all three environments from one application, without needing to SSH into each machine separately to check on container status or make changes.

Portainer helped me build a deeper understanding of how Docker management works; things like network bridges defined in Compose YAML files, container resource settings, volume mounts, and log inspection. Portainer does have a commercial side and the interface can feel cluttered at first with features you may never use, but with time I was able to identify the parts that mattered and work efficiently within it to maintain my homelab applications.

Access Control & Identity Management

Security was at the forefront of this project — an idea that sparked a huge part of the build, and the topic I enjoyed most during graduate school. In the Foundation chapter, the focus was around access control at the firewall level, making sure that only specific devices could reach the homelab and that only the necessary ports were open internally for devices to communicate between VLANs. Many of the applications I started deploying had their own basic authentication built in, and for the most part that's more than enough — especially since I'm the only one logging in. However, I wanted to explore Identity Providers and Single Sign-On, and see what it would take to connect all my apps under one umbrella.

I did a lot of research on which solution was the most secure, approachable to learn, and the right fit for my use case. Each option had its own strengths and weaknesses, but I landed on Authentik. Mostly because of its clean UI and the vast library of pre-built integrations it offered. I also follow Jim's Garage on YouTube, and he featured it in several videos. While Authentik seemed more straightforward than some alternatives, it was also very granular. Despite being open source, it's built with enterprise-scale deployments in mind, so the amount of configuration involved was extensive. I appreciated the detailed logs and the depth of data available for troubleshooting. The one downside I experienced was around updates — they could occasionally break things, so I learned to always take a backup before attempting any version upgrade.

Once I had Authentik set up and ready, it was time to learn how to actually use it. Through the available documentation and tutorial videos, I was able to get a solid understanding of the concepts and setup process. I learned the distinction between a *provider* and an *application* in Authentik's model, and the various rules and settings that control authentication behavior. For most applications, I aimed to enable passwordless authentication. Around 70% of the apps I run on my homelab connect via OIDC (OpenID Connect), which is a modern identity layer built on top of OAuth 2.0. The setup is relatively straightforward once you understand the flow. For the remaining 30%, the applications were simpler and had no native SSO support. For those, Authentik offered a Proxy authentication option, where Authentik injects an authentication layer in front of the app through my reverse proxy. Any time someone tries to access one of those applications, they're redirected to Authentik first. However, some applications had no way to disable their own built-in basic authentication, which meant the proxy pass-through didn't work cleanly — at least not in my environment.

Other applications did support disabling their native login entirely and forwarding authentication straight to SSO, which made the experience seamless. Once you're authenticated, you're in, no second prompt. The logs have been useful for both initial setup and ongoing troubleshooting, though I haven't yet wired Authentik into my email for alert notifications or connected it to a

centralized SIEM for log aggregation — both things on my list.

Setting up my own IdP and establishing SSO across my homelab not only made access easier and more consistent, it gave me hands-on experience with identity and access management concepts that I wouldn't have gotten any other way. It turned out to be one of the first homelab skills I directly applied at work. At the time, my organization was standing up one of its first cloud systems, and the security team wanted to use an IdP rather than exposing access directly to Active Directory. They went with Okta, which operates on very similar principles to Authentik. Since no one on the team had much Okta experience, I was able to step in and help with the initial setup — establishing the connections between Okta, our Active Directory, and various applications. While the corporate environment used SAML 2.0 rather than OIDC, the underlying concepts were familiar enough that I could navigate it confidently. Over time, the number of systems connected through Okta continued to grow.

For the most part I still use Authentik today, but I've started evaluating alternatives. Authentik has a strong community and is a genuinely capable tool, but it comes with more configuration surface area than I actually need, and complexity, when not managed carefully, can itself become a security liability. I began exploring other open-source options like Zitadel and Pocket ID, and ultimately landed on Pocket ID for its simplicity and fully passwordless design. It supports only passkeys, which makes it inherently resistant to brute force and credential stuffing attacks. It's still a relatively new project, and I'm still experimenting to see whether it can fully replace Authentik in my environment. It doesn't have the same depth of logging, and it only supports OIDC, but unlike Authentik, it's extremely lightweight. The entire application runs in a single container, whereas Authentik requires at least four, including a dedicated database, largely because it needs to store credential data. Pocket ID, storing no passwords at all, simply doesn't need that overhead.

This is part of a broader practice I've adopted: every tool in my homelab goes through an annual review to evaluate whether there's something better, more stable, or more actively maintained available in the open-source community. Nothing is set in stone, the homelab evolves as the ecosystem does.

Application Sprawl

Applications

Once the foundation and all the supporting systems were in place, I thought — this is it, now I can actually start running applications. Up until that point, the only non-administrative thing running was my SearXNG search engine. The whole idea of a homelab is partly to self-host applications and tools you'd otherwise pay for or hand off to a cloud provider, but it's also a space for learning and expanding your skills. Most of what I'd built up to this point was infrastructure I *needed* in order to run things — now it was finally time to plug in the rest.

Administration & Monitoring

These applications run primarily on the Overseer VM and form the operational backbone of the homelab — the tools that keep everything else visible, manageable, and healthy.

Started with:

- **Authentik** — Identity Provider used for OIDC-based Single Sign-On across most applications, with proxy authentication configured for apps that don't natively support SSO.
- **NGINX Proxy Manager** — Reverse proxy with DNS-01 challenge support, enabling valid TLS certificates for internal-only domains across all my services, without exposing anything to the public internet.
- **Uptime Kuma** — Lightweight monitoring and notification platform for health checks across services, servers, and other homelab systems. It sends alerts the moment something goes down or becomes unreachable.
- **Dockeep** — A simple application port monitor for keeping track of which ports are in use across the environment, useful for avoiding conflicts as the number of containers grows.
- **Portainer** — Docker container management UI that gave me a visual interface for managing containers across multiple VMs from one place. Over time this was replaced by Komodo, and eventually by Dockge, as I found each to be a better fit for how my workflow had evolved.

Later added:

- **Termix** — A multi-platform server management interface that consolidates SSH terminal access, remote desktop control via RDP and VNC, SSH tunneling, and remote file management into a single web-based UI. It replaced the need to juggle multiple tools for reaching different systems.

- **Nutify** — A UPS monitoring and management platform built on top of Network UPS Tools (NUT). It provides real-time power status, historical telemetry, configurable alerts, and interactive charts through a modern web interface — essential for knowing when power events occur and how long battery runtime remains.
- **Linux Update Dash** — A simple dashboard for tracking and managing pending system updates across all Linux servers in the homelab, making it easy to stay on top of patch status without SSHing into each machine individually.
- **Forgejo** — A self-hosted Git repository platform, used for version-controlling Docker Compose stacks, configuration files, and scripts. Keeps everything backed up and tracked locally without relying on GitHub.
- **Pangolin** — A zero-trust tunneling service that allows secure remote access to internal homelab resources without port forwarding. Used for exposing private services over WireGuard-based tunnels through a lightweight agent called Newt.

Documentation & Knowledge Management

Good documentation is the cornerstone of a successful homelab — especially as the number of services grows and the details of past setups start to blur together.

- **BookStack** — A wiki-style documentation platform organized around books, chapters, and pages. This is where I keep runbooks, setup notes, network diagrams, and anything else I'll need to reference later.
- **Karakeep** — A bookmark management platform for saving and organizing useful websites, guides, and resources I want to return to. Replaces browser bookmarks with something searchable, tagged, and accessible from anywhere.
- **Paperless-ngx** — A document management system for ingesting, OCR-processing, tagging, and searching scanned documents. Later replaced with Papra, which better fit my workflow for organizing personal records and paperwork.

Productivity & Utilities

These are the applications I've found genuinely useful as self-hosted alternatives to proprietary or cloud-based tools — covering everything from diagramming and task management to media and file conversion.

- **Draw.io** — A web-based diagramming tool for creating network diagrams, flowcharts, and architecture visuals. Used regularly for documenting homelab topology.
- **IT Tools** — A collection of handy developer and sysadmin utilities in a single web interface — things like hash generators, encoders, formatters, and network calculators. A surprisingly useful all-in-one toolbox.
- **Bento PDF Editor** — A full-featured PDF editor with 20+ tools for annotating, merging, splitting, compressing, and manipulating PDFs, comparable to what Adobe Acrobat offers but fully self-hosted.

- **Stirling Image** — An image processing platform with 30+ tools for editing, converting, and batch-processing images, similar in scope to Adobe Photoshop and Bridge but running entirely locally.
- **Vikunja** — A project management and task tracking platform. Used for organizing notes and keeping track of ongoing projects across the homelab and beyond.
- **Mini QR Code** — A simple, self-hosted QR code generator. Small utility, but it comes up more often than you'd expect.
- **Transmute** — A privacy-focused file conversion platform that handles images, video, audio, documents, spreadsheets, subtitles, and fonts — all processed locally without uploading anything to an external service.
- **ConvertX** — A complementary file conversion tool with broad format support. Where Transmute covers most conversions, ConvertX fills the gaps.
- **Enclosed** — A minimalist web application for sending self-destructing, encrypted notes and files. Useful for sharing sensitive information without it living in email or a chat log indefinitely.
- **Gharmonize** — A media processing tool for downloading content from multiple platforms. Useful for archiving videos and audio locally for offline access.
- **Jellyfin** — An open-source media server for streaming a personal library of movies and TV shows, similar to Netflix but entirely under your control. No subscriptions, no licensing restrictions, no tracking.
- **Umami** — A privacy-respecting web analytics platform similar to Google Analytics, but without the data collection or third-party tracking. Used for monitoring traffic on self-hosted web properties.
- **Notifuse** — A self-hosted email platform for sending newsletters and transactional emails. A fraction of the cost of services like Mailchimp or SendGrid, with full control over deliverability settings.
- **Gramps Web** — A self-hosted genealogy and family tree platform, serving as a private alternative to Ancestry.com. All family history data stays local.

All self-hosted applications run in Docker containers, managed through whichever container management tool is current at the time. Most were discovered through YouTube channels focused on self-hosting, or through community newsletters like selfh.st, which curates new and updated open-source projects on a regular basis.

It's worth noting that this list represents what stuck. There are many more applications I tried along the way that either failed to deploy cleanly, didn't perform the way I expected, or simply didn't find a purpose in my workflow. That trial-and-error process is part of the homelab experience — and half the learning happens in the ones that don't work out.

Docker Management

When I first started running applications in my homelab, I was managing everything through the command line. Spinning up a container meant crafting a `docker run` command with all the right flags, ports, volumes, environment variables, network bridges, and hoping you remembered every piece of it the next time you needed to touch it. For a container or two, it works fine. But as the number of applications grew, managing everything through CLI became increasingly difficult to keep organized. There was no easy way to see what was running at a glance, restart something quickly, or remember what flags a container had been started with six weeks ago.

That's what led me to **Portainer**.

Portainer

Portainer was a significant quality-of-life improvement. It gave me a visual interface for managing containers across multiple VMs from one place, with the main instance running on Overseer and lightweight agents deployed on the other VMs. I could see running containers, check logs, restart services, and deploy new stacks without touching the command line. For someone still getting comfortable with Docker, having that visual layer made a real difference. The learning curve was manageable, and the interface was intuitive enough to navigate without much friction.

That said, Portainer had its share of limitations. Updates were occasionally painful and required care to avoid breaking the setup. More notably, while it is open source, a significant portion of its more useful features are locked behind a paid Business license. For a homelab running on free and open-source software wherever possible, that always felt like a ceiling I kept bumping into. It worked, but it wasn't the long-term answer.

About a year into the homelab, I came across **Komodo**.

Komodo

Komodo was a different experience from the moment I opened it. The design was clean and modern, the interface was more intuitive, and it handled Docker Compose stacks in a way that finally felt right. Rather than treating stacks as an afterthought, Komodo put them front and center, easy to create, easy to edit, and easy to manage across multiple environments. The feature that really stood out, though, was built-in version control for stacks. Being able to track changes to Compose files, see what changed and when, and roll back if something broke was exactly the kind of workflow I hadn't realized I was missing.

Transitioning from Portainer to Komodo wasn't entirely painless. Docker managers aren't just passive viewers, they actively manage network bridges, container labels, and other configuration details that don't always transfer cleanly. Getting everything migrated across all my VMs and

configured correctly in Komodo took some time and careful attention, but once it was done, the day-to-day experience was noticeably better. Having VM snapshots, was a critical part of this process.

Then, at the beginning of 2026, I found **Dockhand** — and made the switch again.

Dockhand

On the surface, the idea is similar: a clean interface for managing Docker containers and Compose stacks across multiple environments. But there were a few specific things that made Dockhand stand out in practice. Container information, logs, and the built-in terminal are all surfaced much more directly — less clicking to get to what you actually need. The integrated file manager was the detail that pushed me over the edge. Being able to browse and edit files within a container's environment directly from the interface, without SSHing in separately, is a small thing that saves a surprising amount of time.

Around this same period, I also started taking Git more seriously. I'd had Forgejo deployed for a while, but it was mostly sitting idle. Komodo's version control feature had introduced me to the concept, but it was a self-contained system — I wasn't really thinking in terms of a proper Git workflow. When it came time to make the switch to Dockhand, I decided to use the transition as an opportunity to do it properly. I migrated all my Docker Compose stacks over to Forgejo, organized them into repositories, and set up Dockhand to pull from those repos for deployments.

The difference was immediate. Now every stack has a real commit history. Changes are tracked, documented, and reversible. If a deployment goes wrong, rolling back is a matter of seconds rather than trying to remember what the file looked like before you edited it. Dockhand is currently connected across six environments, a mix of VMs, LXC containers, and a cloud server — and deploying or updating a stack anywhere in that environment is fast and straightforward.

The one consistent limitation with both Komodo and Dockhand is resource monitoring. Neither tool reports CPU and RAM usage in a way that lines up consistently with what Proxmox shows at the hypervisor level. It's not a dealbreaker, but it does mean I keep an eye on resource usage from the Proxmox side rather than relying on the Docker manager for that. For everything else though, the current setup, Forgejo for version control, Dockhand for management, is the most capable and organized the homelab has ever been.

Backups & Disaster Recovery

First lesson that every homelab journey eventually teaches, it's that it's not a matter of *if* something will break, it's *when*. Especially if you use used hardware from your old computer or cheap mini pc of ebay. Having a solid backup strategy in place before that moment arrives is the difference between a minor inconvenience and losing months of work. This is an area I've been deliberate about, and one that has already proven its worth in a very real way.

Proxmox Backup Server

The backbone of my backup strategy is a dedicated machine running **Proxmox Backup Server (PBS)**. Rather than relying on the Proxmox VE nodes themselves to manage backups, PBS is a standalone system connected to all my PVE environments — giving me a centralized, independent target for all backup jobs.

Every VM and LXC container in the homelab runs on a daily backup schedule. PBS uses a chunk-based, deduplicated storage format, which means it doesn't store a full copy of every VM every day. Instead, it only captures what has changed since the last backup, and deduplicates redundant data across the entire datastore. The result is that daily backups of a full homelab take a fraction of the storage you'd expect. Beyond the scheduled backups, I also take manual snapshots before any major update or significant work on a VM, a habit that has saved me more than once when an update didn't go as planned.

PBS also runs verification jobs on a regular basis. This is something that often gets overlooked — a backup that can't be restored isn't really a backup. Verification ensures the stored data is intact and recoverable, not just that the job completed without an error.

The real test of any backup system isn't the setup — it's what happens when you actually need it. That moment came recently when one of my mini PCs failed completely, taking an entire PVE environment down with it. I installed Proxmox fresh on a replacement machine, connected it to PBS, and restored the VM server from the last captured backup in under a minute. The restored VM came back with its original configuration intact, including its IP address and MAC address, which meant the firewall rules and network settings required virtually no adjustment. What could have been a multi-hour rebuild turned into a minor interruption. That experience validated the entire strategy.

Off-Site and Secondary Storage

PBS on a single machine is a strong first layer, but it's still a single point of failure. To add another layer of resilience, I connected PBS to my **Synology NAS** as a secondary backup target. Once a week, PBS syncs the last ten backup sets to the Synology, giving me a secondary copy that survives even if the PBS machine itself were to fail.

The Synology adds its own layers of protection on top of that. It runs its own snapshot schedule, has replication capabilities, and maintains a copy to a dedicated physical storage drive. This creates a layered approach where each tier covers the failure scenario of the one above it, PBS covers daily recovery, the Synology covers PBS-level failures, and the physical drive covers the Synology.