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.

Revision #1
Created 2026-03-09 17:07:19 UTC by lumxux
Updated 2026-03-09 17:09:21 UTC by lumxux