Executive summary:

This paper aims to provide a guide on how I built a home lab with recycled hardware. The full setup includes five computers all running various Linux distributions, consisting of a Proxmox cluster of three computers, a network attached storage (nas) like computer running Linux mint, and a main desktop computer running Ubuntu. The goal was to create an environment that's conducive to self-hosting, data science and ML engineering experimentation and general exploration and implementations of many different system architectures. All computers mentioned are past their end-of-life date, thus the firmware and bios no longer receive updates. For security, none of the systems are accessible outside of my firewalled network.

Main desktop:

The main computer acts as a general work and data science experimentation platform. This is where all of the compute heavy workloads are performed and acts as a central programming hub. This section outlines the initial setup and important steps taken to ensure the security and usability of this resource. The desktop itself is a Lenovo ThinkStation S30, with 128GB ram and an Intel Xenon CPU with 6 physical cores (12 virtual cores). The operating system is installed on a 256GB SSD, and for storage two 2TB HDD are configured as a striped (raid0) zfs pool. All data is backed up to an external 4TB HDD on a bi-weekly schedule.

The Operating system for the desktop is Ubuntu 22.04 LTS, with post install steps taken to harden and secure the computer. The first was to update and upgrade the packages through the apt package manager. Second was to make the security updates automatic with the apt package unattended-upgrades, using the default configuration file. This ensures that only security upgrades are automatic. Next, I configured the ssh server, to enable login from my laptop. To do so the sshd_config file was modified to listen on a different port than the default port 22, and disallowance the root user to login via SSH. The ability to login using a password was also disallowed after copying the public rsa key from my laptop to the authorized_keys file on the desktop. Next the uncomplicated firewall (ufw) was enabled which is a wrapper around the iptables interface. The initial desktop settings were changed to only allow in the port that I chose for ssh. Finally, apparmor-profiles was installed and every profile was changed to enforce. These steps represent a baseline for security and stands as a template for further work.

As this computer acts mainly as a workstation very few extra applications were installed. There are four main applications that will be in use regularly. Syncthing, docker, miniconda and vscode. Syncthing is used to synchronize certain files and folders between the desktop, the laptop and the nas. The was setup and configured to start at launch. The docker engine was installed using the official instructions from the docker website (1). A warning that is particularly important here “When you publish a container's ports using Docker, traffic to and from that container gets diverted before it goes through the ufw firewall settings. Docker routes container traffic in the nat table, which means that packets are diverted before it reaches the INPUT and OUTPUT chains that ufw uses. Packets are routed before the firewall rules can be applied, effectively ignoring your firewall configuration.” (“Packet Filtering and Firewalls”). Because of this we must be sure that any ports we expose from docker containers are not allowed in from the external firewall located on the router access point.

My data science work is mainly implemented in python for experimentation and testing. As such robust python versioning environments were required to ensure each project could be maintained using a python environment with specific package versions. Here the anaconda package was ideal as the command line interface (cli) tool conda can be used to create these environments and keep them segregated with different versions of python depending on the needs of the project. More specifically miniconda(2) was used as it only contains the conda cli tool and leaves out the gui and extra packages that come with a traditional anaconda install.

Finally, vscode(3), was installed as an IDE for all programming. It has very useful python linting and documenting support as well as support for many other languages. Vscode is also installed on my laptop and allows for the remote access of machines using the ssh protocol. This is especially useful as this allows the use of the compute power of the desktop remotely from my laptop.

An additional step was created to ensure access to this computer power if the vscode connection failed. This is in the form of a modified docker image customized to my use case. The original docker image was made from an open-source version of vscode called code-server. This was created and maintained by the linuxserver(4) community. I added a layer on top of their base image to include helpful extensions, and to have a prebuilt installation of all the miniconda python environments. This new image allowed the use of the compute power of the desktop in the containerized format that can be accessed through the network without the need for vscode or ssh.

NAS:

The nas itself consists of an old desktop computer from 2009, this is where all files are synchronized using a docker implementation if syncthing. This synchronizes files from the desktop and laptop keeping everything up to date. The computer itself has an AMD FX-6100 CPU with 3 physical cores (6 virtual cores) and 10GB ram. The operating system is installed on a 500GB SSD, and it holds five 4TB HDDs that are configured in a raidz1 (raid5) zfs array with one disk fault tolerance creating a usable zfs pool of 14.3TB.

The os for this nas is Linux Mint 21.3 with an extremely light weight Xfce desktop. The post install steps followed the same as outlined above for the Ubuntu desktop install (except for miniconda and vscode). Additional steps include the implementation of a nfs share for the Proxmox cluster, the details of which are outlined in the section below. This nfs share exports two folders to the static ip's of the proxmox machines and allows those static ip's to access the nfs specific ports through the firewall.

Proxmox Cluster:

The heart of this home lab is the proxmox cluster. It consists of three computers; the first is an Intel NUC with an Intel Core i7-5557U CPU with two physical cores (4 virtual cores) and 16GB ram. The os is installed on a 128GB SSD. The two other computers are Lenovo ThinkCenter M92p tiny's with Intel Core i5-3470T CPU's. One has 16GB ram the other has 8GB ram. For both the os is installed on 500GB HDD's.

The os for all three is Proxmox 8.1 and they are all linked to form the cluster. The resources in total for the cluster combined comes to 12vCPU and 40GB ram. Each node in the cluster has access to two folders from the nfs share of the nas. The first stores all backups and snapshots of vms created on the cluster. The second stores iso images, and vm templates. This second nfs folder also creates a resource for Hi availability within the cluster if the vm is stored in that shared folder. Three nodes are required for high availability at a minimum. This is a key aspect to this cluster as if one node goes down the vm will come up on another node very quickly. The main limitation to this is that the read wright speed of the vm is limited by the network speed, however the high availability trade off makes up for this for specific vms. If fast read wright speed is needed, then the vm will be created on the node with an SSD and backed up to the nfs share in case of a failure.

There are three vms in the cluster that take advantage of high availability. They are all built on a base of Ubuntu server 22.04. The first consists of monitoring services for docker and a lightweight version of kubernetes called k3s (6). It implements services like portainer for monitoring and modifying mainly docker containers and is connected to other docker daemons running on other machines with the portainer agent. For monitoring kubernetes (k3s) a rancher implementation is used on that same vm. The second high availability vm is running Pi-hole (5) which acts as a network wide add blocker and a local DNS service. The third High availability vm is the main node for a kubernetes cluster. There is a worker node on each of the cluster nodes, but the main controller node uses high availability. kubernetes in a normal sense should have three main nodes each with worker nodes to achieve high availability as this is where the controls and database are stored for the worker nodes. However, within the proxmox cluster giving the main node the ability to change cluster nodes if one goes down circumvents this requirement.

Some home lab services that I'm currently running besides the monitoring services are, paperless-ngx(7) which organizes all of my documents and papers and books and creates a searchable database. Plex (8) which organized all movies and tv shows and has some streaming capabilities. Finally, I have a PostgreSQL database that is used for many purposes, but the main one is to track and log data science models that I have created or use locally to keep measurements and performance metrics.

Some upcoming improvements that are in progress are integrating a reverse proxy, ingress controller and load balancer for all services (docker or kubernetes) using traefik(9) and a certificate manager called cert-manager (10). Finally, I am adding a CI CD pipeline using Argo CD (11) to strengthen and document current projects.

Networking:

As it stands today the router hosts the main firewall for the local network. The router connects to an un-managed switch which then connects the cluster, the nas and the desktop. All DNS requests are sent first through pihole from the router, then if it is not defined as a local domain, it is sent upstream to cloud flare's DNS.

For external access to the local network and any service hosted there I currently use an OpenVPN set up on the router, using tunnelblick(7) as the client from the laptop. This is important as the firmware is all past its end-of-life date. Meaning that any security flaws will not be patched going forward.

Future Improvements:

Additional steps for improving the home lab would include the acquisition and implementation of a dedicated firewall that sits between the modem and the rest of the network. The firewall on the router that is implemented now is good, but more fin grained control is necessary. Next is a nas system that can run a version of truenas scale. This would require upgrading to lots of EEC memory and thus a new motherboard and CPU as well. Finally, a new desktop workstation for heavy loads and the existing desktop would then be incorporated into the proxmox cluster. I would keep the current cluster of small machines going for applications requiring high availability, but the addition of the current desktop would provide the extra ram and CPU cores for hosting trained data science models and pipelines for example. For networking I would like to have the nas, proxmox cluster and desktop on a 10 Gigabit managed switch for a fast transfer rate and fast read wright speed for high available vms. Finally, I would like to set up multiple vlans for separate use cases within my network. One, that would be walled off from the rest of the network would be used to host public facing services, another would be used for phones, tablets, TVs, and other household/iot devices and a third would be used for trusted local devices such as the main proxmox cluster the nas and office desktop. Every step when exposing any public facing service must be meticulously thought out and narrowly defined.

Links:

(1) https://docs.docker.com/engine/install/ubuntu/

(2) https://docs.anaconda.com/free/miniconda/#quick-command-line-install
(3) https://code.visualstudio.com/docs/setup/linux
(4) https://www.linuxserver.io/
(5) https://pi-hole.net/
(6) https://docs.k3s.io/
(7) https://docs.paperless-ngx.com/
(8) https://www.plex.tv/
(9) https://traefik.io/traefik/
(10) https://cert-manager.io/
(11) https://argoproj.github.io/cd/
(12) https://tunnelblick.net/

References:

"Packet Filtering and Firewalls." *Docker Documentation*, 11 Jan. 2024, docs.docker.com/network/packet-filtering-firewalls/#docker-and-ufw. Accessed 21 Feb. 2024.