Executive summary:

This paper aims to provide a guide on how I built a home lab with recycled hardware. The full configuration includes five computers all running various Linux distributions. The first group consists of a Proxmox virtual environment cluster of three small form factor machines. The second is a network attached storage (nas) running Truenas Scale, and the third a main desktop workstation running Ubuntu. The goal was to create an environment that's conducive to self hosting, data science, 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. With this in mind 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 a truenas dataset via syncthing. Thus the non redundant nature of the striped array is acceptable. Implementing the array in this way does come with compromises. On one hand it implements a faster read wright speed than a mirrored array and can utilize the full capacity of both disks. On the other hand if one disk fails not only will the disk need to be replaced but all the data will need to be restored from the nas, which would create downtime. However for my specific use case I value the speed and capacity increase this configuration creates.

The Operating system for the desktop is Ubuntu 22.04 LTS, with post install steps taken to harden and secure the computer. The first step is 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. Next I configured the ssh server, to enable login and resource use from my laptop and secure the service. The steps included configuring the sshd_config file to listen on a different port than the default port 22, and disallowed the root user to login via ssh entirely. 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 were 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 application were installed. There are four main applications that will be in use regularly. Syncthing, docker, miniconda and vscode. Syncthing(1) is used to sync certain files and folders between the desktop, the laptop and the nas. This was setup and configured to start at launch. The Docker(2) engine was installed using the official instructions from the docker website. 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 python environment manager 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(3) 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(4), 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 it allows me to use the compute power of the desktop remotely from my laptop.

An additional step was created to ensure access to this compute power if vscode for one reason or another 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. It is created and maintained by the linuxserver(5) community. I added a layer on top of the their base image to include helpful extensions, and to have a prebuilt installation of all the miniconda python environments I use regularly. 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. One downside to this was the size of the image itself. However as it will only sit on the workstation and will rarely be recreated I felt it was an acceptable trade off for a backup plan.

NAS:

The nas itself consists of an old desktop computer from 2009 and an external hdd drive case from a dead 8-bay synology nas DS1815+. The compute spects include 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 both drive bays combined hold 13 4TB HDDs that are configured in two data vdevs each using raidz2. Thus each vdev one of 8 disks and one with 5 disks each have a fault tolerance of two disks. This crates about 32TB of usable storage.

The os for this nas is Truenas scale(6), which is a build on debian. The older truenas core which was built on OpenBSD although seemingly lighter had trouble starting on this specific hardware. The system supports apps which are all configures in k3s(7), a lightweight kubernetes distribution. This is prebuilt into truenas and is the default way to use apps on this platform. As there is limited ram I wanted to reserve the bulk of it for zfs, thus the only app in use here is Syncthing.

Both of the data vdevs are combined in the same zfs pool, which contains all of the datasets used. Permissions and credentials are set up for smb shares across the local network, for apps within the system and for ssh access to the system. Each role has specific permissions and controls associated with their access.

To ensure data is backed up from both the workstation and the laptop, I have implemented two distinct stratagies. The first is Syncthing to synchronize data between the three devices. The second is snapshot replication that stores the most recent snapshots from the ubuntu workstation, and a time machine backup from the mac. Replication has also been set up on the truenas system enabling a rollback to an earlier state if necessary.

Proxmox Cluster:

The heart of this home lab is the proxmox(8) cluster. It consists of three computers; the first is an Intel NUC with a Intel Core i7-5557U CPU with two physical cores (4 virtual cores) and 16GB ram. 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 all three the os is installed on 1TB HDD’s.

The os installed is Proxmox v8.1 and they are all linked to form a cluster. The cluster can be seen and managed from any of the three web UI's. The resources in total for the cluster combined comes to 12vCPU and 40GB ram. Each node in the cluster has access to two nfs share located on 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 disk is stored in that shared folder. An odd number of nodes with a minimum of three are required for high availability as a "quorum" vote is necessary to determine where the resources are transferred if a node drops. 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, at the moment a max of 1Gbps, however the high availability trade off makes up for this deficiency in critical applications such as the pi-hole(9) local DNS. All vms are backed up to the nfs share daily using snapshots to enable quick rollback of accidental changes or data corruption.

There are two vms in the cluster that take advantage of high availability. They are built on a base of Ubuntu server 22.04. The first consists of monitoring services for docker and a light weight version of kubernetes called k3s. This vm implements services like portainer(10) for monitoring and modifying mainly docker containers and compose projects. This is connected to other docker daemons running on other machines with the portainer agent. For monitoring kubernetes (k3s) a rancher(11) implementation is used on that same vm. The second high availability vm is running pi-hole which acts a network wide add blocker and a local DNS service.

Within the proxmox cluster I have set up a k3s kubernetes cluster made up of three controller node vms. The base of these are all an Ubuntu server 22.04 cloud image created as a template using cloud init. This greatly simplifies the process of configuring each vms from scratch as I can set the username, password and upload the public ssh keys of my workstation and laptop as part of the template. These three are then customized and set up further with an Ansible(12) playbook to make the process of creating the cluster streamlined and repeatable. The cluster is then initialized with Metallb(13) as a load balancer that can be reached from outside the cluster.

Some home lab services that I'm currently running besides the monitoring services are, paperless-ngx(14) which organizes all of my documents, papers and books, and creates a searchable database. Plex(15) which has some free media and live tv streaming services. Finally I have a PostgreSQL(16) database that is used for many purposes but the main one is to track and log data science models that I have created and for use locally to keep measurements and performance metrics. These are all implemented as docker compose projects.

Finally to compliment the pihole local DNS I have implemented a traefik(17) reverse proxy and a certificate manager called cert-manager(18) in the kubernetes cluster which keeps track of and automatically renews certifications from letsencrypt for my local services. This is plainly overkill as my network is not publicly facing, however I was tired of seeing self signed certificate warnings.

Some upcoming improvements that are in progress are, integrating a CI CD pipeline using Argo CD(19) to strengthen and document current projects. I would also like to implement more monitoring solutions like grafana(20), uptime-kuma(21), home-assistant(22), and a homepage where I can access all of the services in one location using a docker app called homepage(23).

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 cloudflare's DNS.

For external access to the local network and any service hosted there I currently use an OpenVPN set up on the router, accessing it using the tunnelblick(24) client from the laptop. This is important as the firmware on all computers except the router is all past their 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 I would like to upgrade the nas to newer hardware. 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, TV's, 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 taken when exposing any public facing service must be meticulously thought out and narrowly defined.

Links:

(1) https://syncthing.net/
(2) https://docs.docker.com/engine/install/ubuntu/
(3) https://docs.anaconda.com/free/miniconda/#quick-command-line-install
(4) https://code.visualstudio.com/docs/setup/linux
(5) https://www.linuxserver.io/
(6) https://www.truenas.com/truenas-scale/
(7) https://docs.k3s.io/
(8) https://www.proxmox.com/en/
(9) https://pi-hole.net/
(10) https://www.portainer.io/
(11) https://www.rancher.com/
(12)https://www.ansible.com/
(13) https://metallb.org/
(14) https://docs.paperless-ngx.com/
(15) https://www.plex.tv/
(16) https://www.postgresql.org/
(17) https://traefik.io/traefik/
(18) https://cert-manager.io/
(19) https://argoproj.github.io/cd/
(20) https://grafana.com/
(21) https://github.com/louislam/uptime-kuma
(22) https://www.home-assistant.io/
(23) https://github.com/gethomepage/homepage
(24) https://tunnelblick.net/

Refrences:

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