Scot's Ubuntu NAS Documentation

Release 0.1.0beta

Scot W. Stevenson

Jun 15, 2019

Contents:

CHAPTER 1

Introduction

In 2019, the motherboard of my FreeNAS server died. Instead of simply replacing it and continuing, I decided to switch to a self-created NAS based on Ubuntu Server 18.04 LTS with ZFS and containers (Docker). This document is based on the line-by line notes I made, and provided for anybody who is thinking about building such a system themselves.

However, **you probably don't want to do this**. I'm no server expert, this machine is set up for a very special use case and threat scenario, and this text doesn't even attempt to provide a complete guide. One reason I wrote it, in fact, is to learn reStructured text, Sphinx, and Read the Docs for other projects.

Still, if you feel you must, by all means read on.

> **Warning:** Follow the instructions and suggestions in this text at your own risk. There is a very real chance that you could lose data, even all of it. I take no responsibility for anything bad or even mildly irritating that happens.

If you need more hand-holding: FreeNAS (http://freenas.org) comes with extensive documentation and a very helpful community. It served me well for years, and if I hadn't being trying to learn more about servers and containers, I would have stayed with it. FreeNAS comes with the ZFS file system as well.

If you need *lots* more hand-holding: There are a bunch of **commercial NAS** vendors, for example Synology or QNAP. Usually, you just need to add the drive and the setup does the rest; however, they rarely come with ZFS.

If you need more power and features: Ansible-NAS (https://github.com/davestephens/ansible-nas) is a far more ambitious variant of a Ubuntu NAS by Dave Stephens. As the name says, it makes extensive use of Ansible for configuration, and comes with a higher level of configuration. Full disclosure: I have contributed to the documentation.

The actual files for this text live at https://github.com/scotws/ubuntu-nas. For suggestions, corrections, and additions, please create a pull request there.

CHAPTER 2

Goals

This introduces the basic setup in the house where the NAS will be installed, the other machines on the network, and some of the names and addresses used.

Note: Some of this information differs from the actual setup for security reasons. It is possible that this has created inconsistencies at some points when I didn't pay attention. Please report these as errors.

2.1 House overview

The NAS is part of a computer setup in a single-family house with four human users and two cats. The local network is not accessible from the outside. Internet connection is through a DSL modem, which also (currently) serves as the DHCP server. The local DNS connection goes through PiHole (https://pi-hole.net/) on a Raspberry Pi.

There are also Windows computers in the house, mainly for gaming, but they are considered **untrusted** by default and may not access the NAS.

Various people work at various strange times in the house, so the Raspberry Pi and the NAS itself are on all the time. Also, cats have weird sleep schedules.

2.2 Network

To keep things simple, all machines will be considered to be part of the IPv4 local network 192.168.13.0/24. For the purpose of this document, the server's name will be **home** and the main computer we work from **chell**, after the character in the Valve computer game. Most machines are assigned *static addresses*. In particular:

home	192.168.13.2	NAS	Ubuntu Server
chell	192.168.13.20	user PC	Ubuntu Desktop
worker	192.168.13.21	user PC	Ubuntu Desktop
mediator	192.168.13.22	laptop	MacOS

There are also various other machines such as mobile phones, Chromebooks and iPads that connect to the network but are not relevant here.

In practice, most user addressing uses mDNS (zero configuration) internally. As we will see later, this means we access the NAS with commands such as

```
ssh home.local
```

This has the pleasant side effect of automatically locking out the Windows machines, which do not support zero configuration out of the box.

2.3 NAS duties

The NAS in this house is expected to provide the following services, in order of importance:

- Storage for documents and media, especially family photos
- Auto backup for the users' data from Linux and MacOS machines (not: Windows)
- Low-traffic Emby server (usually only one user)
- Serve NFS shares to one other Ubuntu machine chell
- Allow experimenting with containers and virtual machines

In the future, other functions might be added such as game server or a family chat bot.

The storage function has the highest priority, especially for family photos, which are considered irreplaceable. As such, they must be protected from bitrot by a *self-healing file system*. In practice, this means either ZFS or Btrfs. ZFS has been in production longer, and Btrfs is far less battle-tested and comes with various warnings about how things are (still) not production ready. Therefore, ZFS is the main mass storage file system for the NAS.

Important: At the most basic level, this NAS is a life-support system for ZFS.

The other functions are secondary.

2.4 Threat assessment

The NAS is operating in a low-threat environment. It is not accessible from the internet, there is especially no VPN or other (known) way to reach it from outside. The machine itself is physically secured in the sense that if bad people are in my house, the threat to the NAS will not be my first concern. Still, the setup will follow *best practices* as far as possible with hardening and firewall.

The main threat here is **data loss** through hard drive failure, power outage, cat intervention, or other such factors. The main strategy here is *multiple redundant backups*, some of them off-site.

The secondary threat is **user screw-ups**, especially accidentally deleting data.

2.5 NAS overview

Though the NAS can be accessed directly by the console, mostly it will be administered from the computer chell. This functions as a **jump server** - other machines will be barred from accessing the NAS as much as possible. Since chell is powered down when not in use, this means that accessing the NAS through the network using ssh is usually not possible at all.

CHAPTER 3

Hardware

Most of the hardware was taken from the previous FreeNAS install.

3.1 Motherboard

The main difference is the new motherboard, a Supermicro X10SDV-4C-7TP4F Out of the box, you can attach **20 SATA 3.0 drives** and install **one M.2 NVMe PCIe 3.0** memory stick. This is currently not used, as are the **two SFP+ 10 GBit Ethernet** ports. They are for later expansions.

The processor is an **Intel Xeon D-1518 4/8 core 2.20 GHz**, which is overkill for current use but should be powerful enough for later virtual machines.

3.2 Drives

In this first incarnation, the **root file system** for Ubuntu 18.04 LTS is on a 120 GB Intel 540S SSD. Root on ZFS is still too complicated for this project.

Note: At time of writing, ZFS on Linux 0.8 was not yet integrated to the Ubuntu kernel, this is using 0.7.9-3ubuntu6

The **mass storage** was inherited from the FreeNAS build and consists of one ZFS pool ("tank") constructed from two RaidZ VDEVs in 3/3/3 TB and 4/4/4 TB configuration. The total size is 14 TB on six drives.

3.3 Extra cooling fan

The X10SDV-4C-7TP4F is shipped without a CPU cooler, which doesn't work. Instead of buying an extra cooler and replacing the passive heat sink, we install a jury-rigged fan that blows air on the heat sink.

3.4 Other hardware

RAM: 2 x 8 GB DDR4 ECC RAM This is left over from a different project. As finances allow, these will be replaces by 4 x 16 GB ECC RAM for a total of 64 GB. Though ZFS loves RAM and the maximum for the board is 128 GB, this should be enough for our use.

Graphics: VGA built-in In contrast to the usual setup with servers, there is an old 4:3 monitor attached with VGA. Also, there is keyboard. Both are in the same room as the main computer. This means we can do stuff directly at the console which otherwise would require ssh.

Case: Fractal Design Define R5 White FD-CA-DEF-R5-WT Chosen for the soundproofing, provides easy storage for eight HDs and two SSDs.

UPS: APC Back-UPS 700VA BX700U-GR

Power: Seasonic SSR-450RM Active PFC G-450 (450W, ATX 12V)

3.5 IPMI

The motherboard comes with IPMI enabled through either a separate network interface or shared with on of the normal 1 GB Ethernet connections. We really don't need it, but it's there and we have to secure it.

In our set, IPMI gets an address at boot via DHCP, for example something like 192.168.13.100. Write it down from the boot screen and then when the server is powered up, use a web browser to access the IPMI interface.

The most important thing is to change the default Supermicro user name and password from `ADMIN/ADMIN` to something different.

3.5.1 Links

https://www.servethehome.com/basic-bmc-and-ipmi-management-security-practices/

3.6 BIOS

Make sure that **virtualization support** is enabled. Also, the **boot order** can be tricky to get right.

3.7 LSI interface

The 16 SATA port interface card will show its own boot screen during startup. We do not need to change anything. Note the X10SDV-4C-7TP4F only includes the "dumb" IT mode, so we don't have to flash the BIOS to avoid hardware RAID configurations which just get in ZFS' way.

CHAPTER 4

Ubuntu Server

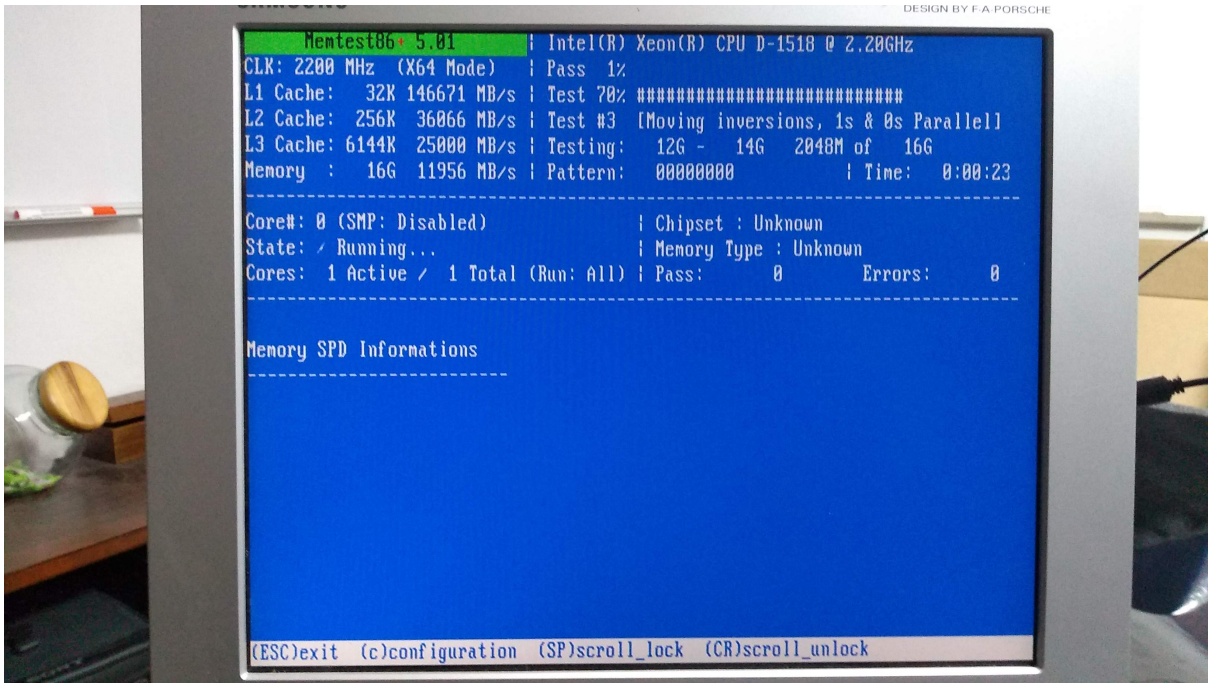
We use Ubuntu Server 18.04 LTS which is supported until 2023. We download it the normal way from https://www.ubuntu.com/download/server . We check to make sure that the file is intact by running

```
echo "ea6ccb5b57813908c006f42f7ac8eaa4fc603883a2d07876cf9ed74610ba2f53 *ubuntu-18.
↪04.2-live-server-amd64.iso" | sha256sum --check
```

which gives you an OK. Move to USB stick with *Startup Disk Creator*.

4.1 Memory Test

Before we can proceed, we use the Ubuntu USB stick's memory test function to make sure that our RAM is okay.



This can take hours.

4.2 Basic installation

After the memory test, reboot and start installing the server.

Warning: During the install, do not select the option to install Docker. This will install Docker into a snap, which will then not work correctly. In fact, you probably want to avoid all snaps.

Afterwards, we update the system and reboot to be sure:

```
sudo apt update
sudo apt upgrade
sudo reboot
```

The name of the user we install as will be `user1` in our examples.

Note: Most of the examples here have `sudo` in front of them to mark that they have to be executed by the superuser. If you tire of this (and you will), use

```
sudo su -
```

to switch to root permanently. Remember to `exit` as soon as possible.

CHAPTER 5

Livepatch

Canonical Livepatch automatically updates some parts of the kernel without rebooting.

Log into https://ubuntu.com/livepatch and get the id number. On the machine,

```
sudo snap install canonical-livepatch
sudo canonical-livepatch enable <NUMBER>
canonical-livepatch status --verbose
```

We don't do this on chell because we reboot the thing all the time anyway.

5.1 Links

- http://blog.dustinkirkland.com/2016/10/canonical-livepatch.html

CHAPTER 6

Services

6.1 mDNS (Avahi)

Zero configuration, also known as mDNS or Avahi on Linux, is not installed by default with Ubuntu Server. This means we can't access `.local` addresses. This is our primary main way of talking to machines. We will want to do this via the console.

```
sudo apt install avahi-daemon
sudo apt install avahi-utils
```

Note we will have to punch a hole in the firewall later for `UDP 5353` for local machines.

To get a list of machines on the network, use

```
avahi-browse -a
```

6.1.1 Links

- https://kb.iweb.com/hc/en-us/articles/360005117952-Guide-to-Multicast-DNS-mDNS-security-issues

6.2 Network

We want home to have a static address. In fact, our network is small and unchanging enough we can use static addresses for a lot of things.

```
sudo vi /etc/netplan/50-cloud-init.yaml
```

We change this to:

```
network:
    ethernets:
        eno1:
            addresses: [192.168.13.2/24]
            gateway4: 192.168.13.1
            nameservers:
```

(continues on next page)

(continued from previous page)

```
            addresses: [192.168.13.8,8.8.8.8]
        dhcp4: no
    eno2:
        dhcp4: true
    eno7:
        dhcp4: true
    eno8:
        dhcp4: true
version: 2
```

Despite the scary text in the file, this survives reboots every time. Here, restart the network. From the terminal, execute

```
sudo netplan apply
```

In the router, make sure that we get the same address every time via DHCP for this machine.

6.2.1 Links

- https://websiteforstudents.com/configure-static-ip-addresses-on-ubuntu-18-04-beta/
- https://www.ostechnix.com/how-to-configure-ip-address-in-ubuntu-18-04-lts/

6.3 SSH

Though we use the console for lots of stuff, in practice, we'll want to be able to access the server via ssh. Remember chell is the jump server for home, other machines should not be able to access it.

We change a bunch of options to "harden" the server. We start by editing `/etc/ssh/sshd_config`:

```
AllowUsers user1
ClientAliveCountMax 0
ClientAliveInterval 300
IgnoreRhosts yes
LoginGraceTime 2m
MaxAuthTries 5
MaxSessions 3
PermitEmptyPasswords no
PermitRootLogin no
Port 2019
PrintMotd yes
Protocol 2
X11Forwarding no
```

Note that we change the default port (to the year this was written). We'll go over some of these later when we take a look at the firewall. Then restart the ssh demon.

```
sudo systemctl restart ssdh
```

We don't want to let anybody log in with just their password, instead, we need them to have *public keys* generated on chell. We will then copy it over from there to home. Don't use a passphrase when prompted:

```
ssh-keygen
cat .ssh/id_rsa.pub
ssh-copy-id -p 2019 -i ~/.ssh/id_rsa.pub user1@home.local
```

On home, we then edit `/etc/sshd_config` to include:

```
PubkeyAuthentication yes
PasswordAuthentication no
```

Restart again. Because logging in with a different port and all of that gets old fast, we create a file `~/.ssh/config` on home with the content:

```
Host home
    Hostname home.local
    User user1
    Port 2019
```

Now we can just login from chell with `ssh home` as user1. We cannot just log in via password. Remember chell is the jump server for home.

6.3.1 Links

- https://linux-audit.com/audit-and-harden-your-ssh-configuration/
- https://linux-audit.com/using-ssh-keys-instead-of-passwords/)

CHAPTER 7

Users

We have four users on the system, here we'll name them *user1* to *user4*. We already have user1 from setting up the operating system. For NFS, we have to make sure that the UID and GID are the same on chell and home.

```
sudo adduser user2
sudo adduser user3
sudo adduser user4
```

This gives us:

```
user1   1000:1000
user2   1001:1001
user3   1002:1002
user4   1003:1003
```

Add group home with GUID 1010 (used for pictures):

```
sudo groupadd -g 1010 home

sudo adduser user1 home
sudo adduser user2 home
sudo adduser user3 home
sudo adduser user4 home
```

The cats do not need separate user accounts, they just use root when they feel like it.

CHAPTER 8

Mail

We need a basic mail setup to send notifications from ZFS and other systems in case of an error. We use Postfix.

```
sudo apt install postfix
```

During setup, configure for an "Internet Site". We use `home.local` as the sending address and `smtp.gmail.com:587` as the relay host. Go to https://myaccount.google.com/apppasswords to get a password for "chell mail" and create the file `/etc/postfix/sasl/sasl_passwd` with the content:

```
[smtp.gmail.com]:587 <USERNAME>@gmail.com:<PASSWORD>
```

This needs to be added to a data bank:

```
sudo postmap /etc/postfix/sasl/sasl_passwd
```

In `/etc/postfix/main.cf` set the line

```
relayhost = [smtp.gmail.com]:587
```

Shorten the line with the banner to

```
smtpd_banner = $myhostname ESMTP
```

for security reasons. And add the bunch of lines

```
# Enable SASL authentication
smtp_sasl_auth_enable = yes
# Disallow methods that allow anonymous authentication
smtp_sasl_security_options = noanonymous
# Location of sasl_passwd
smtp_sasl_password_maps = hash:/etc/postfix/sasl/sasl_passwd
# Enable STARTTLS encryption
smtp_tls_security_level = encrypt
# Location of CA certificates
smtp_tls_CAfile = /etc/ssl/certs/ca-certificates.crt
```

Edit aliases file with `sudo vi /etc/aliases` so it reads:

```
postmaster: root
admin: root
ubuntu: root
root: <USERNAME>@gmail.com
```

The run sudo newaliases for the databank file and sudo systemctl restart postfix. Test with a mail that ends with a dot:

```
sendmail <USERNAME>@gmail.com
From: <USERNAME>@<OTHER_ADDRESS>
Subject: Test mail
This is a test email
```

This is basic mail setup. We need the mailutils package for further use.

```
sudo apt install mailutils
```

Note: This setup will now allow us to send all kinds of mail with

```
mail -s "<SUBJECT>" the.address@the.address
<MORE TEXT>
<CTRL>-d
```

8.1 Links

- https://www.linode.com/docs/email/postfix/configure-postfix-to-send-mail-using-gmail-and-google-apps-on-debian-or-ubur

CHAPTER 9

SMART

Installing the SMART disk drive monitoring system. This assumes email notifications are set up correctly.

```
sudo apt install smartmontools
```

Edit `/etc/default/smartmontools` so that `start_smartd=yes`. Now edit `/etc/smartd.conf` with a first test setup of

```
DEVICESCAN -m <USERNAME>@gmail.com -M test
```

Then restart the service with

```
sudo service smartd restart
```

Should send an email for every drive. Then edit the line to show

```
DEVICESCAN -a -s (S/../../1/19|L/../02/./21) -m <USERNAME>@gmail.com
```

Where `-a` is `-H -f -t -l error -l selftest -C 197 -U 198`. This does a short test every Monday at 19:00h and a long test every second day of the month at 21:00h - make sure there isn't a ZFS scrub at the same time.

```
sudo service smartd restart
```

And restart the service again.

Note: NVMe SMART support is rudimentary, see https://www.smartmontools.org/wiki/NVMe_Support.

9.1 Links

- https://help.ubuntu.com/community/Smartmontools
- https://wiki.archlinux.org/index.php/S.M.A.R.T.
- https://www.smartmontools.org/browser/trunk/smartmontools/smartd.conf.5.in

CHAPTER 10

UPS

This is with the UPS APC 700VA BX700U-GR

```
sudo apt install apcupsd
sudo cp /etc/apcupsd/apcupsd.conf /etc/apcupsd/apcupsd.conf.bak
sudo vi /etc/apcupsd/apcupsd.conf
```

Notes some of these are just the defaults.

```
ANNOY 300
ANNOYDELAY 60
BATTERYLEVEL 20
DATATIME 0
DEVICE
EVENTSFILE /var/log/apcupsd.events
EVENTSFILEMAX 10
KILLDELAY 0
LOCKFILE /var/lock
LOGSTATS off
MINUTES 10
NETSERVER on
NISIP 127.0.0.1
NISPORT 3551
NOLOGINDIR /etc
NOLOGON disable
ONBATTERYDELAY 6
POLLTIME 90
PWRFAILDIR /etc/apcupsd
SCRIPTDIR /etc/apcupsd
STATFILE /var/log/apcupsd.status
STATTIME 0
TIMEOUT 0
UPSCABLE usb
UPSCLASS standalone
UPSMODE disable
UPSNAME BX700U
UPSTYPE usb
```

We also have to edit

```
sudo vi /etc/default/apcupsd
```

and there set `ISCONFIGURED=yes`. Restart the service after configuration:

```
sudo systemctl restart apcupsd.service
sudo systemctl status apcupsd.service
```

The command sudo apcaccess gives a dump of features. Then to test the settings, stop the demon and run

```
sudo apctest
```

We still have to test it by pulling the plug.

10.1 Links

- http://www.apcupsd.org/manual/manual.html
- https://www.pontikis.net/blog/apc-ups-on-ubuntu-workstation)

CHAPTER 11

ZFS

As described earlier, we use ZFS because it is one of the few file systems that can detect and correct bitrot. At time of writing, Ubuntu is the only major Linux variant to support ZFS out of the box.

11.1 The current setup

Note: We inherited our data storage pool "tank" from the previous FreeNAS install, so there is currently no discussion of how to setup and configure a ZFS pool and datasets.

We import the pool with `zpool import -f tank`. Since this computer will be used as a NAS, we don't limit the size of the ARC, which defaults to half the RAM.

```
sudo install zfsutils
```

We currently have the following file systems ("datasets") for bulk storage:

```
tank/coldstore
tank/media
tank/pictures
tank/storage
tank/texts
```

Also, each of the users has a separate dataset for backups:

```
tank/h_user1
tank/h_user2
tank/h_user3
tank/h_user4
```

Finally, there is a file system for the Time Machine backups:

```
tank/TM_mediator
```

11.2 Email notifications

This assumes we have basic mail notifications working with postfix.

```
sudo vi /etc/zfs/zed.d/zed.rc
```

so that we have (note these are commented out by default):

```
ZED_EMAIL_ADDR="<USER1'S MAIL ADDRESS>"
ZED_EMAIL_PROG="mail"
ZED_EMAIL_OPTS="-s '@SUBJECT@' @ADDRESS@"
ZED_NOTIFY_VERBOSE=1
ZED_NOTIFY_DATA=1
```

Follow this with

```
sudo systemctl restart zed
```

Test with scrub of tank, should send mail.

```
sudo zpool scrub tank
```

We will setup snapshots in a later step.

11.2.1 Links

- https://github.com/zfsonlinux/zfs/issues/6246

CHAPTER 12

Containers

Note: We currently use Docker, though Podman might make more sense for a system this size because it doesn't need a separate demon and doesn't run with root. However, Ubuntu currently has better support for Docker than Podman.

You will remember we didn't install Docker during the setups of the operating system because that would have given us the wrong version.

Note: This is the easy way to install, though it doesn't give you the newest version (a common problem with Ubuntu). See https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04 for instructions on how to install the program from the official Docker repository.

The basic install of Docker is with

```
sudo apt install docker.io
sudo systemctl enable docker
```

The second line is responsible for autostarting. What follows are the individual container installations. We also install a container for Time Machine backups, but we'll get to that later.

12.1 Glances

Glances provides an overview of the system. Though there are monitors that provide more information, this seems to be a good balance of size and function for a NAS with our profile.

```
sudo docker pull nicolargo/glances
```

Create a folder `/root/Docker-scripts`. There, create a file `glances-docker.sh`:

```
docker run -d \
--name glances-home \
--restart unless-stopped \
--publish 61208-61209:61208-61209  \
```

(continues on next page)

(continued from previous page)

```
--env GLANCES_OPT="-w" \
--volume /var/run/docker.sock:/var/run/docker.sock:ro \
--pid host \
docker.io/nicolargo/glances
```

We start with is `sudo ./glances-docker.sh`. To reach the interface, go to http://home.local:61208 .

Note: This might need some tweaking, because Glances is unhappy when 70 percent of memory is used, which is normal for a NAS.

12.2 Emby

Emby is one of three options for TV and movie streaming, the other two are Plex and JellyFin. We create a volume for configuration files, so we don't have to reorganize every time there is a new version.

```
sudo docker pull emby/embyserver:latest
sudo docker volume create emby-config
```

Create a file in `/root/Docker-scripts/` named `emby-docker.sh` with the content:

```
docker run -d \
--name emby-home \
--volume emby-config:/config \
--volume /tank/media/movies:/mnt/movies:ro \
--volume /tank/media/series:/mnt/tv:ro \
--publish 8096:8096 \
--publish 8920:8920 \
--restart unless-stopped \
--env UID=1000 \
--env GID=1000 \
emby/embyserver:latest
```

The actual media files are kept in the ZFS datasets `tank/media`, and we pass them `ro` (read-only) to be paranoid. Run it and setup at http://home.local:8096 . During setup, set thread count to 4 for the moment.

12.2.1 Configuration backups

We keep a backup of the `emby-config` volume from `/var/lib/docker/volumes/emby-config` at `/tank/storage/BKU Emby/`. The uncompressed size (02. June 2019) is about 3 GB.

12.3 Watchtower

Watchtower is a program to automatically update other containers. This seems a bit overkill for our small collection, but we might as well set it up.

```
sudo docker pull v2tec/watchtower
```

Create `watchtower-docker.sh` file in the usual folder, configured to run once a night:

```
docker run -d \
--name watchtower-home \
-v /var/run/docker.sock:/var/run/docker.sock \
v2tec/watchtower --schedule "0 0 5 * * *" --cleanup
```

This runs the check once a day at five in the morning and gets rid of old stuff once it has updated.

12.4 Time Machine

We do this with Docker because Ubuntu (again) uses an ancient version of netatalk.

```
docker run -dit \
  --name timemachine-home
  --restart=unless-stopped
  -v /tank/TM_mediator:/timemachine
  -p 548:548
  -p 636:636
  --ulimit nofile=65536:65536
odarriba/timemachine
```

We have to run another line:

```
docker exec timemachine-home add-account user1 <PASSWORD> Hive-TM /timemachine
```

On the Mac, use COMMAND-K to input `afp://192.168.13.2/Hive-TM` which is the name of the share. This could probably be home.local as well. Mount the drive that way by hand and then configure it as a new Time Machine backup drive.

12.5 Links

- https://hub.docker.com/r/v2tec/watchtower/
- https://godoc.org/github.com/robfig/cron#hdr-CRON_Expression_Format
- https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04
- https://github.com/nicolargo/glances
- https://glances.readthedocs.io/en/stable/docker.html
- https://glances.readthedocs.io/en/stable/cmds.html#interactive-commands
- https://hub.docker.com/r/odarriba/timemachine/
- https://github.com/odarriba/docker-timemachine

CHAPTER 13

NFS

Only operating systems from the Unix family - Linux, MacOS - will be accessing this NAS and the environment is relatively secure, so we can use NFS instead of Samba for better performance.

```
sudo apt install nfs-kernel-server
```

We link the various datasets through a folder so we don't have to export the actual datasets. This is experimental.

```
mkdir /exports

ln -s /tank/pictures/ /exports/pictures
ln -s /tank/coldstore/ /exports/coldstore
ln -s /tank/media/ /exports/media
ln -s /tank/storage/ /exports/storage
ln -s /tank/texts/ /exports/texts
```

In `/etc/exports`, add the following lines:

Note: We leave the addresses as 192.168.0.0 instead of 192.168.13.0 as might be expected because we'll be putting the high-speed SFP+ network on a different subnet at some point.

```
/exports *(ro,fsid=0,no_subtree_check)
/exports/pictures 192.168.0.0/255.255.0.0(rw,no_subtree_check)
/exports/coldstore 192.168.0.0/255.255.0.0(rw,no_subtree_check)
/exports/media 192.168.0.0/255.255.0.0(rw,no_subtree_check)
/exports/storage 192.168.0.0/255.255.0.0(rw,no_subtree_check)
/exports/texts 192.168.0.0/255.255.0.0(rw,no_subtree_check)
```

On the target machine, we need the setup in `/etc/fstab`:

```
home.local:/exports/coldstore /mnt/coldstore  nfs noatime,noauto 0 0
home.local:/exports/storage   /mnt/storage    nfs noatime,auto 0 0
home.local:/exports/media     /mnt/media      nfs noatime,auto 0 0
home.local:/exports/pictures  /mnt/pictures   nfs noatime,auto 0 0
```

Then run `sudo exportfs -a` to make sure we know about this. Finally, just to be sure, try

```
sudo systemctl restart nfs-kernel-server
```

13.1 Links

- https://unix.stackexchange.com/questions/106122/mount-nfs-access-denied-by-server-while-mounting-on-ubuntu-machines

CHAPTER 14

Snapshots

We use Sanoid https://github.com/jimsalterjrs/sanoid to automatically create and prune ZFS snapshots. Note this is a backup server, so we don't need hourly snapshots for the users because we're only fed their stuff once a day anyway.

Sanoid is not part of the Ubuntu distribution, so we need to set it up from the GitHub repository.

```
sudo apt install libconfig-inifiles-perl
cd /opt
sudo git clone https://github.com/jimsalterjrs/sanoid
sudo ln /opt/sanoid/sanoid /usr/sbin/
sudo mkdir /etc/sanoid
sudo cp /opt/sanoid/sanoid.conf /etc/sanoid/sanoid.conf
sudo cp /opt/sanoid/sanoid.defaults.conf /etc/sanoid/sanoid.defaults.conf
```

In `/etc/sanoid.conf` we have the templates and the datasets they refer to.

```
[tank/coldstore]
        use_template = archive

[tank/h_user3]
        use_template = slacker

[tank/h_user2]
        use_template = backup

[tank/h_user1]
        use_template = busybee

[tank/h_user4]
        use_template = slacker

[tank/media]
        use_template = archive

[tank/pictures]
        use_template = archive

[tank/storage]
        use_template = archive
```

(continues on next page)

(continued from previous page)

```
[tank/texts]
        use_template = archive


### TEMPLATES ####

# User who only occasionally logs in. We want to be able to catch when they
# come the next day and says "dude, I lost my file"
[template_slacker]
        frequently = 0
        hourly = 0
        daily = 7
        weekly = 4
        monthly = 4
        yearly = 1
        autosnap = yes
        autoprune = yes

# User who does a lot of work and could lose a lot
[template_busybee]
        frequently = 0
        hourly = 0
        daily = 31
        weekly = 0
        monthly = 12
        yearly = 1
        autosnap = yes
        autoprune = yes

# Backup for people whose data comes from outside and is transferred by rsync.
[template_backup]
        frequently = 0
        hourly = 0
        daily = 7
        weekly = 4
        monthly = 12
        yearly = 1
        autosnap = yes
        autoprune = yes

# Backup for media files and other archived data. We usually don't
# delete anything in these datasets, but only add stuff, so we want
# to avoid cryptolocker attacks and catch "oops" accidents.
[template_archive]
        frequently = 0
        hourly = 0
        daily = 3
        weekly = 1
        monthly = 3
        yearly = 0
        autosnap = yes
        autoprune = yes
```

Edit `/etc/crontab` by hand (we don't care about skipping one hour during daylight savings transition):

```
*/5 * * * * root /usr/sbin/sanoid --cron
```

After about five minutes, you'll see snapshots appearing when you run `zfs list -t snapshot`.

Note: The datasets used for Time Backups are not snapshotted.

14.1 Deleting snapshots

Use the `-n` switch to test what we do before we do it. Format is

```
zfs destroy -vn <FIRST-SNAPSHOT>%<LAST-SNAPSHOT>
```

Where last snapshot is only the part after the `@`. For example:

```
zfs destroy -vn tank/h_user1@auto-20190303.0900-3m%auto-20190601.0900-3m
```

This will also tell you how much space will be freed. Do this with without `-vn` to pull the trigger.

CHAPTER 15

Rsync

We use rsync for backups from the Linux computers to the NAS.

Note: This is a rather crude way of doing things, left over from earlier setups. In future versions, this will probably be replaced by `zfs send/receive` or Syncoid (https://github.com/jimsalterjrs/sanoid)

In the user file `/home/user1/rsync_home.sh` we put:

```
#!/bin/bash
rsync -az -e 'ssh -p 2019' --delete --exclude={.cache,.steam,.zfs,.dbus,go,snap} /
↪home/user1/ user1@home.local:/tank/h_user1/bku_chell
```

We can test this with the rsync `vn` flags to be certain. Then, add crontab job with `crontab -e`:

```
05 17  * * * /home/user1/rsync_home.sh
```

For the other users, we need to generate ssh keys, such as user2. Then, on home as that user:

```
mkdir .ssh
vi .ssh/authorized_keys
```

Copy the public key `id_rsa.pub` content to that file. To test, run

```
ssh home.local -p 2019
```

once from user2's account. Then, test as above (with nv) options:

```
rsync -az -e 'ssh -p 2019' --delete --exclude={.cache,.steam,.zfs,.dbus,snap} /
↪home/user2/ user2@home.local:/tank/h_user2/bku_chell
```

Note: This line is the reason we do not issue a banner with sshd.

If that works, put it into a shell script like for user1 with a slightly different time:

```
25 17 * * * /home/user2/rsync_home.sh
```

Repeat the process with other users.

15.1 Links

- https://www.linode.com/docs/security/authentication/use-public-key-authentication-with-ssh/
- https://jrs-s.net/2016/09/15/zfs-snapshots-and-cold-storage/
- https://blog.fosketts.net/2016/08/18/migrating-data-zfs-send-receive/

CHAPTER 16

Firewall

We have the advantage that there is an actual terminal hooked up to the server, so we can just walk over and configure stuff by hand if it doesn't work. In other words, we can be very restrictive. The goals:

1. Everybody can access **Emby**
2. The Apple computers can access **Time Machine**
3. Everybody on the local net can access **Glances**
4. Core is the jump server for home, so chell can **ssh**; worker needs to ssh into home for backups
5. We need to allow access to Avahi (**mDNS**) to all in the local network
6. Allow **NFS** access from chell

Out of the box, we get an "inactive" with

```
sudo ufw status
```

Which is good. First we start with the basics:

```
sudo ufw default deny incoming
sudo ufw default allow outgoing
```

Then we explicitly reject ssh via the normal port 22 (don't leave the user hanging, because this will be a simple mistake that will be made often):

```
sudo ufw reject ssh
```

For the other goals:

Goal 1: Emby

```
sudo ufw allow to any port 8096 comment 'Emby HTTP'
```

Goal 2: Time Machine access from mediator

```
sudo ufw allow from 192.168.13.22 to any port 548 comment 'TM from mediator'
sudo ufw allow from 192.168.13.22 to any port 427 comment 'TM from mediator'
```

Goal 3: Glances from local network

```
sudo ufw allow from 192.168.13.0/24 to any port 61208 comment 'Glances'
```

Goal 4: Allow ssh from chell and worker

```
sudo ufw allow proto tcp from 192.168.13.20 to any port 2019 comment 'ssh from␣
↪chell'
sudo ufw allow proto tcp from 192.168.13.21 to any port 2019 comment 'ssh from␣
↪worker'
```

Goal 5: Allow mDNS

```
sudo ufw allow mdns comment 'mDNS'
```

Goal 6: Allow NFS access from chell This is more tricky, because the mountd changes the port by default.

Edit `/etc/default/nfs-kernel-server` and change the line

```
RPCMOUNTDOPTS="--manage-gids"
```

and make it into

```
RPCMOUNTDOPTS="--port 33333"
```

This is just a number that was easy to remember. Now restart the server.

```
sudo service nfs-kernel-server restart
```

Add the correct firewall rules:

```
sudo ufw allow from 192.168.13.20 to any port nfs comment 'NFS from chell'
sudo ufw allow from 192.168.13.20 to any port 111 comment 'NFS from chell'
sudo ufw allow from 192.168.13.20 to any port 33333 comment 'NFS from chell'
```

Disable and re-enable the firewall.

Status verbose gives us:

```
To            Action      From
--            ------      ----
5353          ALLOW IN    Anywhere           # mDNS
2019/tcp      ALLOW IN    192.168.13.20      # ssh from chell
2019/tcp      ALLOW IN    192.168.13.21      # ssh from worker
8096          ALLOW IN    Anywhere           # Emby HTTP
548           ALLOW IN    192.168.13.22      # Time Machine from mediator
427           ALLOW IN    192.168.13.22      # Time Machine from mediator
61208         ALLOW IN    192.168.13.0/24    # Glances
22/tcp        REJECT IN   Anywhere           # No SSH over normal port
2049          ALLOW IN    192.168.13.20      # NFS from chell
111           ALLOW IN    192.168.13.20      # NFS from chell
33333         ALLOW IN    192.168.13.20      # NFS from chell
5353 (v6)     ALLOW IN    Anywhere (v6)      # Allow Zero Config
8096 (v6)     ALLOW IN    Anywhere (v6)      # Emby HTTP
22/tcp (v6)   REJECT IN   Anywhere (v6)      # No SSH over normal port
```

Administration stuff that might come in handy:

```
sudo ufw enable           # start the firewall
sudo ufw status verbose   # what's going on
sudo ufw app list         # who can pierce the firewall
sudo iptables -L          # list of rules
sudo ufw disable          # stop the firewall
```

Test with various machines to see if we can log in / do time machine / play videos.

16.1 Links

- https://www.cyberciti.biz/faq/how-to-setup-a-ufw-firewall-on-ubuntu-18-04-lts-server/
- https://www.digitalocean.com/community/tutorials/how-to-set-up-a-firewall-with-ufw-on-ubuntu-18-04
- https://help.ubuntu.com/community/UFW
- https://wiki.ubuntu.com/UncomplicatedFirewall
- http://manpages.ubuntu.com/manpages/bionic/en/man8/ufw.8.html
- https://wiki.debian.org/SecuringNFS
- https://serverfault.com/questions/377170/which-ports-do-i-need-to-open-in-the-firewall-to-use-nfs-

CHAPTER 17

Tests

We use Lynis to look for security holes. Unfortunately, Ubuntu (again) contains an old version. The following is how get the newest version:

```
dpkg -s apt-transport-https | grep -i status
```

Shows us that the HTTPS transport is not installed. Do this with

```
sudo apt-get install apt-transport-https
```

Now that we can use HTTPS we can install the program itself.

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys␣
↪C80E383C3DE9F082E01391A0366C67DE91CA5D5F
echo "deb https://packages.cisofy.com/community/lynis/deb/ stable main" | sudo tee␣
↪/etc/apt/sources.list.d/cisofy-lynis.list
sudo apt update
sudo apt install lynis
```

Finally we're ready to rock:

```
sudo lynis audit system | less -R
```

Walk through the output and figure out how to secure stuff.

```
cat /var/log/lynis.log | grep Suggestion
```

17.1 Links

- https://packages.cisofy.com/community/#debian-ubuntu
- https://www.digitalocean.com/community/tutorials/how-to-perform-security-audits-with-lynis-on-ubuntu-16-04
- https://cisofy.com/lynis/controls/

CHAPTER 18

Speed

These are tests performed with a 1 GBit Ethernet connection. To test the connection speed with iperf, install it first on both machines, chell and home.

```
sudo apt install iperf3
```

On home, start the server (`-f M` gives format in MB/sec)

```
iperf3 -s -f M
```

Now iperf will tell us which port it is listening at. We have to open the firewall for testing:

```
sudo ufw allow 5201
```

On chell, start the client with :

```
iperf3 -f M -c home.local
```

This gives us **112 MB/sec** for memory-to-memory transfers, as expected over a 1 GBit Ethernet connection. Disk to memory test gives us the same thing, with this on chell (the sender):

```
iperf3 -f M -c home.local -i1 -t 40
```

For memory-to-disk, we start the server on home (with CWD as `/home/user1` on an SSD)

```
iperf3 -s -f M -F test
```

the receiving end. This gives us about **13.4 MB/sec** from memory to a SSD. We can also test with storing the data on one of the ZFS pool datasets (currently 2 x RaidZ three-disk VDEV) :

```
iperf3 -s -f M -F /tank/h_user1/test
```

This gives us **9.7 MB/sec** from memory to HD RaidZ.

On home, remember to close the firewall again and to delete the test files,

```
sudo ufw deny 5201
rm test
rm /tank/h_user1/test
```

18.1 Summary

Type	Network	Speed
mem-to-mem	1 GBit	112 MB/sec
mem-to-ssd (ext4)	1 GBit	13.4 MB/sec
mem-to-hd (ZFS)	1 GBit	9.7 MB/sec

18.2 Links

- https://fasterdata.es.net/performance-testing/network-troubleshooting-tools/iperf/disk-testing-using-iperf/

CHAPTER 19

Indices and tables

- genindex
- modindex
- search