Raspberry Pi 4 - Persistent system acting as a NAS and Time Machine

Tinker, Tailor, Raspberry Pi

I went ahead and got myself a Raspberry Pi 4B with 4GB RAM, which I intend to use as a job scheduling server, only to find out that the suggested OS, Raspberry Pi OS, is 32-bit. Fortunately, the Linux distro Alpine, which I've grown very fond of lately, is available for Raspberry Pi as aarch64, meaning it's both 64-bit kernel and userland. Unfortunately the distro is currently, as of version 3.12, not set up for persistent storage and is more of a live playground. Gathering bits and pieces from various guides online, this can however be remedied with some tinkering. On this page you will find how to set up a persistent 64-bit OS on the Raspberry Pi, share a USB attached disk, while also adding some interesting software.

If you go ahead and buy the Pi 4, note that it has micro-HDMI ports. I thought they were mini, for which I already had cabling, but alas, another adapter had to be purchased. Also, when attaching a USB disk it is better if it is externally powered. The Pi can however power newer external SSD drives that have low power consumption. I tried with a magnetic disk based one powered over USB first, but it behaved somewhat strangely. With that said, let's go ahead and look at how to get yourself a shiny tiny new server.

Tinkering for Persistence

After downloading the v3.12 tarball from Alpine on my macOS, it's time to set up the SDHC card for the Pi. I actually borrowed my old hand-me-down MacBook Air that I gave to my daughter a few years ago, since it has a built-in card reader, as opposed to my newer Air. The Pi boots off a FAT32 partition, but we want the system to reside in an ext4 partition later, so we will start by reserving a small portion of the card for the boot partition. This is done using Terminal in macOS with the following commands.

```
diskutil list
diskutil partitionDisk /dev/disk2 MBR "FAT32" ALP 256MB "Free Space" SYS R
sudo fdisk -e /dev/disk2
> f 1
> w
> exit
```

The tarball should have decompressed once it hit your download folder. If not, use the option "xvzf" for tar.

```
cd /Volumes/ALP
tar xvf ~/Downloads/alpine-rpi-3.12.0-aarch64.tar
nano usercfg.txt
```

The newly created file usercfg.txt should contain the following:

```
enable_uart=1
```

```
gpu_mem=32
disable_overscan=1
```

The least amount of memory for headless is 32MB. The UART thing is beyond me, but seems to be a recommended setting. Removing overscan gives you more screen estate. If you intend to use this as a desktop computer rather than a headless server you probably want to allot more memory to the GPU and enable sound. Full specification for options can be found on the official Raspberry Pi homepage.

After that we just need to make sure the card is not busy, so we change to a safe directory and thereafter eject the card (making sure that any pending writes are finalized).

```
cd ..
diskutil eject /dev/disk2
```

Put the SDHC card in the Pi and boot. Login with “root” as username and no password. This presumes that you have connected everything else, such as a keyboard and monitor.

```
setup-alpine
```

During setup, select your keymap, hostname, etc, as desired. However, when asked where to store configs, type “none”, and the same for the apk cache directory. If you want to follow this guide to the point, you should also select “chrony” as the NTP client. The most important part here though is to get your network up and running. A full description of the setup programs can be found on the Alpine homepage.

```
apk update
apk upgrade
apk add cfdisk
cfdisk /dev/mmcblk0
```

In cfdisk, select “Free space” and the option “New”. It will suggest using the entire available space, so just press enter, then select the option “primary”, followed by “Write”. Type “yes” to write the partition table to disk, then select “Quit”.

```
apk add e2fsprogs
mkfs.ext4 /dev/mmcblk0p2
mount /dev/mmcblk0p2 /mnt
setup-disk -m sys /mnt
mount -o remount,rw /media/mmcblk0p1
```

Ignore the warnings about extlinux. This and the following trick was found in the Alpine Wiki, but in some confusing order.

```
rm -f /media/mmcblk0p1/boot/*
cd /mnt
rm boot/boot
mv boot/* /media/mmcblk0p1/boot/
rm -Rf boot
mkdir media/mmcblk0p1
ln -s media/mmcblk0p1/boot boot
```

Now the mountpoints need fixing, so run:

```
apk add nano
nano etc/fstab
```

If you prefer some other editor (since people tend to become religious about these things) then feel free to use whatever makes you feel better than nano. Add the following line:

```
/dev/mmcblk0p1   /media/mmcblk0p1   vfat   defaults   0 0
```

Now the kernel needs to know where the root filesystem is.

```
nano /media/mmcblk0p1/cmdline.txt
```

Append the following at the end of the one and only line in the file:

```
root=/dev/mmcblk0p2
```

After exiting nano, it's safe to reboot, so:

```
reboot
```

After rebooting, login using "root" as username, and the password you selected during setup-alpine earlier. Now you have a persistent system and everything that is done will stick, as opposed to how the original distro was configured.

Tailoring for Remote Access

OpenSSH should already be installed, but it will not allow remote root login. We will initially relax this constraint. Last in this article is a section on hardening where we again disallow root login. If you intend to have this box accessible from the Internet I strongly advice on hardening the Pi.

```
nano /etc/ssh/sshd_config
```

Uncomment and change the line (about 30 lines down) with PermitRootLogin to:

```
PermitRootLogin yes
```

Then restart the service:

```
rc-service sshd restart
```

Now you should be able to ssh to your Pi. The following steps are easier when you can cut and paste things into a terminal window. Feeling lucky? Then now is a good time to disconnect your keyboard and monitor.

Keeping the Time

If you selected chrony as your NTP client it may take a long time for it to actually correct the clock. Since the Pi does not have a hardware clock, it's necessary to have time corrected at boot time, so we will change the configuration such that the clock is set if it is more than 60 seconds off during the first 10 lookups.

```
nano /etc/chrony/chrony.conf
```

Add the following line at the bottom of the file.

```
makestep 60 10
```

Check the date, restart the service, and check the (now hopefully corrected) date again.

```
date
rc-service chronyd restart
date
```

Having the correct time is a good thing, particularly when building a job scheduling server.

Silencing the Fan

Together with the Pi I also bought a fan, the Pimoroni Fan Shim. According to reviews it is one of the better ways to cool your Pi, but it's still too soon for me to have an opinion. Unless controller software is installed, it will always run at full speed. It's not noisy, but still noticeable sitting a metric meter from the Pi. Again, some tinkering will be needed since the controller software needs some prerequisites installed. We lost nano between reboots, so we will go ahead and add it again.

```
apk update
apk upgrade
apk add nano
```

Other software we need is in the "community" repositories of Alpine. In order to active that repository we need to edit a file:

```
nano /etc/apk/repositories
```

Uncomment the second line (ending in v3.12/community), exit, then install the necessary packages.

```
apk update
apk add git bash python3 python3-dev py3-pip py3-wheel build-base
```

After those prerequisites are in place, install the fan shim software using:

```
git clone https://github.com/pimoroni/fanshim-python
cd fanshim-python
```

```
./install.sh
apk add py3-psutil
cd examples
./install-service.sh
```

The last script will fail with “systemctl: command not found”, since Alpine uses OpenRC as its init system, and not systemd which this script presumes. We will instead write our own startup script:

```
nano /etc/init.d/fanshim
```

This new file should have the following contents:

```
#!/sbin/openrc-run
name="fanshim"
command="/usr/bin/python3 /root/fanshim-python/examples/automatic.py"
command_args="--on-threshold 65 --off-threshold 55 --delay 2"
pidfile="/var/run/$SVCNAME.pid"
command_background="yes"
```

There are a lot of interesting options for fanshim that you can explore, like tuning it’s RGB led. Now we want this to run at boot time, so add it the the default runlevel, then start it.

```
rc-update add fanshim default
rc-service fanshim start
```

Enjoy the silence!

Adding and Sharing a Disk

Some of files we will be transferring are going to be quite large. It would also be neat to be able to access files easily from the Finder in macOS, so I am adding a USB3 connected hard disk with 4TB storage. What follows will be very similar to setting up a NAS, and in fact, the way I fell in love with Alpine was by building my own NAS from scratch (with the minor differences being more disks and using zfs).

First we need to change the filesystem. The disk comes formatted as FAT32, which is very poorly suited for a networked disk. Samba, which is what we will be using for sharing, more or less requires a filesystem that supports extended attributes. After plugging in the drive, we will therefore repartition the drive and format it to ext4.

```
cfdisk /dev/sda
```

Using cfdisk, delete any existing partitions and create one new partition. It should become “Linux filesystem” by default. Don’t forget to “Write” before “Quit”. Then format it:

```
mkfs.ext4 /dev/sda1
```

Now we need to add autofs to get automatic mounting. This package is in edge/testing though, so we need to enable that branch and repository, but still have main and community take

preference. This can be done by labelling a repository.

```
nano /etc/apk/repositories
```

Change the line with the testing repository (last line in my file) to the following. Note that yours will have some server.from.setup/path depending on what you selected in setup-alpine. You only uncomment and add the @testing label in other words.

```
@testing http://<server.from.setup/path>/edge/testing
```

Now autofs can be installed from the labelled repo.

```
apk add autofs@testing
```

Note that dependencies are still pulled from main/community to the extent it is possible. In order to configure autofs, first:

```
nano /etc/autofs/auto.master
```

Add the following line after the uncommented line starting with /misc. It will also disconnect the hard disk after 5 minutes to save energy:

```
/-   /etc/autofs/auto.hdd   --timeout=300
```

Then create this new config file:

```
nano /etc/autofs/auto.hdd
```

Add the the following line to the empty file.

```
/hdd   -fstype=ext4   :/dev/sda1
```

Now, the user pi needs to be created.

```
adduser pi
smbpasswd -a pi
```

Select desirable passwords for the pi user. The latter one will later be stored in the macOS keychain and therefore easy to forget, so make note of it somewhere.

Add autofs to startup and start it now. Change the ownership of /hdd to pi.

```
rc-update add autofs default
rc-service autofs start
chown -R pi.pi /hdd
```

With that in place (disk can be accessed through /hdd) it is time to set up the sharing. For this

we will use samba and avahi for network discovery.

```
apk add samba avahi dbus
nano /etc/samba/smb.cfg
```

Now, this is what my entire smb.cfg file looks like, with all the tweaks to get stuff running well from macOS.

```
[global]
 create mask = 0664
 directory mask = 0775
 veto files = /.DS_Store/lost+found/
 delete veto files = true
 nt acl support = no
 inherit acls = yes
 ea support = yes
 security = user
 passdb backend = tdbsam
 map to guest = Bad User
 vfs objects = catia fruit streams_xattr recycle
 acl_xattr:ignore system acls = yes
 recycle:repository = .recycle
 recycle:keeptree = yes
 recycle:versions = yes
 fruit:aapl = yes
 fruit:metadata = stream
 fruit:model = MacSamba
 fruit:veto_appledouble = yes
 fruit:posix_rename = yes
 fruit:zero_file_id = yes
 fruit:wipe_intentionally_left_blank_rfork = yes
 fruit:delete_empty_adfiles = yes
 server max protocol = SMB3
 server min protocol = SMB2
 workgroup = WORKGROUP
 server string = NAS
 server role = standalone server
 dns proxy = no
[Harddisk]
 comment = Raspberry Pi Removable Harddisk
 path = /hdd
 browseable = yes
 writable = yes
 spotlight = yes
 valid users = pi
 fruit:resource = xattr
 fruit:time machine = yes
 fruit:advertise_fullsync = true
```

Those last two lines can be removed if you are not interested in using the disk as a Time Machine backup for your Apple devices. I will likely not use it, but since this is how I configured my NAS and it was a hassle to figure out how to get it working I thought I'd leave it here for reference. Doesn't hurt to keep it there in any way.

Let us also configure the avahi-daemon, by creating a config file for the samba service. Avahi will announce the server using Bonjour, making them easily recognizable from macOS (where they automagically show up in the Finder).

```
nano /etc/avahi/services/samba.service
```

This new file should have the following contents:

```
<?xml version="1.0" standalone='no'?>
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
<name replace-wildcards="yes">%h</name>
<service>
<type>_smb._tcp</type>
<port>445</port>
</service>
<service>
<type>_device-info._tcp</type>
<port>0</port>
<txt-record>model=RackMac</txt-record>
</service>
<service>
<type>_adisk._tcp</type>
<txt-record>sys=waMa=0,adVF=0x100</txt-record>
<txt-record>dk0=adVN=HDD,adVF=0x82</txt-record>
</service>
</service-group>
```

Not that the txt-record containing adVN=HDD can be removed if you are not interested in using the disk as a Time Machine backup. Still, leaving it won't hurt.

Finally, it's time to add samba and avahi to the startup and start the services.

```
rc-update add samba default
rc-update add avahi-daemon default
rc-service samba start
rc-service avahi-daemon start
```

The disk should now be visible from macOS. Remember to click "Connect as…" and enter "pi" as the username and your selected smbpasswd from earlier. Check the box "Remember this password in my keychain" for quicker access next time. Sometimes, due to a bug in Catalina, you may get "The original item cannot be found" when accessing the remote disk. If that happens, force quit Finder, and you should be good to go again. If anyone knows of any other fix to this issue, let me know!

Automation

Now, this server will be used as a job server. Some of the jobs running will need the psql command from PostgreSQL and some others will be R jobs. Let's install both, or whatever you need to satisfy your desires. You can skip this step for now if you are undecided about what to run or just need basic services like the built-in shell scripting.

```
apk add R postgresql
```

In order to automate these jobs, we will be using Cronicle. It depends on node.js so we need to install the prerequisites. It's run script is fetched using curl, so it will also need to be installed.

```
apk add nodejs npm curl
```

The installation is done as follows (it is a oneliner even if it looks broken here).

```
curl -s https://raw.githubusercontent.com/jhuckaby/Cronicle/master/bin/install.js | node
```

I want to use standard ports, so I need to change the config slightly.

```
nano /opt/cronicle/conf/config.json
```

Change base_app_url from port 3012 to 80. Much further down, change http_port from 3012 to 80, and https_port from 3013 to 443. If you want mails to be sent, change smtp_hostname in the beginning of the file to the mail relay you are using. After that an initialization script needs to be run.

```
/opt/cronicle/bin/control.sh setup
```

Now we just need to get it running at boot time. This is, however, a service that we do not want to "kill" using a PID, so we are going to enable local scripts that start and stop the service in a controlled manner instead.

```
rc-update add local default
nano /etc/local.d/cronicle.start
```

This new file should have the following line in it:

```
/opt/cronicle/bin/control.sh start
```

Now we need to create a stop file as well:

```
nano /etc/local.d/cronicle.stop
```

This file should have the contents:

```
/opt/cronicle/bin/control.sh stop
```

In order for the local script daemon to run these, they need to be executable.

```
chmod +x /etc/local.d/cronicle.*
```

With that, let's secure things.

Hardening

Now that most configuring is done, it's time to harden the Pi. First we will install a firewall with some basic login protection using the builtin 'limit' in iptables. Assuming you are in the 192.168.1.0/24 range, which was set during setup-alpine, the following should be run. Only clients on the local network are allowed access to shared folders.

```
apk add ufw@testing
```

```
rc-update add ufw default
ufw allow 22
ufw limit 22/tcp
ufw allow 80
ufw allow 443
ufw allow from 192.168.1.0/24 to any app CIFS
ufw allow Bonjour
```

With the rules in place, it's time to disallow root login over ssh, and make sure that only fresh protocols are used.

```
nano /etc/ssh/sshd_config
```

Change the line that previously said yes to no, and add the other lines at the bottom of the file (borrowed from this security site):

```
PermitRootLogin no
PrintMotd no
Protocol 2
HostKey /etc/ssh/ssh_host_ed25519_key
HostKey /etc/ssh/ssh_host_rsa_key
KexAlgorithms curve25519-sha256@libssh.org,diffie-hellman-group-exchange-sha256
Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr
MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-512,hmac-sha2-256,umac-128@openssh.com
```

After that, enable ufw and restart sshd. Note that if something goes wrong here you will need to plug in a monitor and keyboard again to login locally and fix things.

```
ufw enable
rc-service sshd restart
```

Now is a good time to reboot and reconnect to check that everything is working.

```
reboot
```

With root not being able to login, you will instead login as "pi". It is possible for this user to (temporarily, until exit) elevate privileges by the following command:

```
su
```

Another option is to use sudo, but I will leave it like this for now, and go ahead with setting up some jobs. That's a story for another article though.

I hope this guide has been of help. It should be of use for anyone tinkering with Alpine on their Raspberries, and likely some parts for those running other Linux flavors on different hardware as well.
