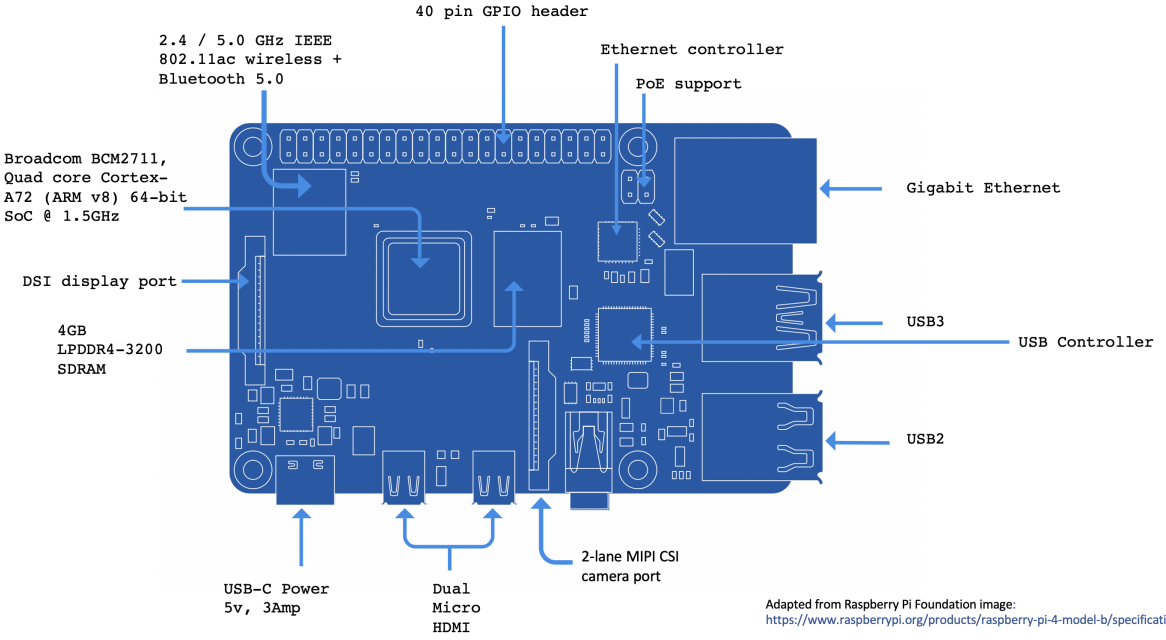
RaspberryPi Recipes

Release 0.0.1

Apr 14, 2023

Introduction

Raspberry Pi 4 – Model B



Adapted from Raspberry Pi Foundation image:
https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/

CHAPTER 1

Introduction

This is the documentation for a collection of recipes, code and hacks for the raspberry pi.

1.1 What are the objectives

Show how the relatively inexpensive Raspberry Pi Single Board Computer (SCB) can be used to learn and/or reinforce understanding of a wide range of topics relating to computer architechure, networking, security, physical computing, programming, and related topics.

1.2 Why the Raspberry Pi?

The cost of entry is low allowing for the purchase dozens of boards in order to experiment and build with. The cost of error is also low. Worse case a computer costing $50 will be ruined. A misconfiguered device with settings that cannot be rolled-back is easy to recover from as the SD card that a raspberry-pi uses for storage of both its OS and data can be wiped and another attempt made.

Experimenting and tinkering with a with a full price computer, installing new operating systems, software, or even a driver, can ruin an existing set-up resulting in hours or days attempting to recover and hopefully fixing the problem.

1.3 Why the elaborate documentation?

The purpose is to document the objectives, sequence of steps, and the outcome in a way that is memorialized for future reference. I want everything to be replicable.

In doing so I'll hopefully improve how I document things and learn about ReStructuredText (.rst) markup syntax for technical documentation.

CHAPTER 2

Raspberry Pi initial setup

Unless purchased as part of a "product bundle" each raspberry-pi arrives as a single-board-computer. Nothing else. Each board requires:

- Micro SD card (class 10, UHS-1 for faster speed. 16gb or 32gb)
- Power supply (see here)
- Optional: keyboard, mouse, monitor.

2.1 Flash Operating System to the SD-card

The "Rasbian" operating system which is based on the debian linux distribution and provided by the Raspberry Pi Foundation works fine for most purposes.

The Raspberry Pi Foundation now provides a utility for choosing the OS and then imaging directly to the SD-Card. Alternatively for a little more control:

1) Download and save in a folder the latest Rasbian images from here. The full version includes a GUI for the OS and other software. The lite version which does not come with a GUI nor other software will suffice when accessing the pi from the command line over ssh.
2) Flash the image onto an SD Card using etcher for Mac OS.
3) In the boot partition create an empty file to enable ssh

```
sudo diskutil mount /dev/disk2s1  #or whatever the boot partition is

cd /volumes/boot

touch ssh
```

4) Enable connection to wifi by creating a `wpa_supplicant.conf` file in the same boot partition:

```
nano wpa_supplicant.conf
```

and enter the following information:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=US

network={
    ssid="NETWORK-NAME"
    psk="NETWORK-PASSWORD"
    key_mgmt=WPA-PSK
}
```

Save the file and exit. Unmount and eject the SD card and insert into the Raspberry Pi and power on.

2.2 Additional OS configurations

1) Access and login to the raspi over wifi:
   - Login into the router portal (eg) `http://192.168.1.1` in order to determine which IP address the router has assigned to the new device that is now connected to the network. This can also be done using a network scanner. Then ssh into the pi:

     ```
     ssh pi@192.168.1.186
     ```

   - Alternatively, if the raspberry pi is the only one on the network (or at least the only one that is still has the default hostname `raspberrypi` then you can access more generically with:

     ```
     ssh pi@raspberrypi.local
     ```

Update the OS and other programs

2) Likely not needed any more but to be on the safe side expand the file-system to take advantage of the SD-card capacity:

   ```
   sudo raspi-config --expand-rootfs
   ```

3) Update various configurations via command line via `sudo raspi-config`:
   - password
   - set the locale
   - update timezone
   - set a hostname (eg rasp-4a)
   - enable vnc

If the light version of the OS is installed or the raspi is *only* ever going to be used via the command-line as a headless device then the gpu memory allocation can be reduced to the 16mb minimum. Set via `advanced options` in `raspi-config`, or directly in the boot config file:

```
sudo nano /boot/config.txt
```

and add the following line at the bottom: `gpu_mem=16`

4) Install any linux command-line utilities and programs as needed. eg to install `screen`, basic calculator `bc` etc.

```
sudo apt-get install screen
sudo apt-get install bc
```

2.3 Enable VNC access

In the same way that SSH (Secure Shell) allows access to the command line of the raspberry pi, VNC (Virtual Networking Computing) allows access to the GUI of the raspberry pi if/when needed.

1) Enable VNC via `` `sudo raspi-config` `` if not already done.
2) Set a password via:

```
sudo vncpasswd -service

#should return "Successfully set password VNC parameter in /root/.vnc/config.d/
↪vncserver-x11"
```

3) Create the following file containing a single line:

```
sudo nano /etc/vnc/config.d/common.custom

Authentication=VncAuth
```

Then restart the vnc service:

```
sudo systemctl restart vncserver-x11-serviced
```

4) You may need to re-enable vnc via `raspi-config` and you may also need to install and run `tightvncserver`.

When accessing the desktop UI remotely we first need to `sudo raspi-config` and set resolution to something that makes sense depending on the client (usually the highest resolution for a mac) and also set `boot to desktop`.

From a safari browser the navigate to `vnc://192.168.1.184` and enter password when prompted.

2.4 SSH key and public key

Access to the raspberry pi via SSH from within your network is usually safe, but if access is desired from outside the local network (ie internet) then a mere username + password combination may be vulnerable.

Security can be enhanced with the use of SSH key and public key for authentication. More information on the SSH protocal, keys etc can be found here.

1) In the home ‘‘~‘‘directory of the mac client machine generate the key, set a passphrase for the private key, and copy the public key to the pi

```
ssh-keygen

ssh-copy-id pi@192.168.1.184
```

If the private key is likly to be one of many then rename it. Delete the public key or achive it somewhere.

```
mv id_rsa raspi4a

rm id_rsa.pub
```

You may need to `chmod 600 raspi4a` for correct permissioning.

2) On the raspberry-pi server update various configuration by opening:

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

Uncomment/enable `PubkeyAuthentication yes`. Set the `Port` number to match port-forwarding on the router, and make sure enable `PasswordAuthentication no`.

3) Log-in to the network router/switch and enable port-fowarding with the port number (avoid the common Port 22…) mapped to the raspberry pi.

4) Restart the service with `sudo service ssh restart`

5) Confirm that the selected port is listerning: `netstat -tnl`

CHAPTER 3

How this documentation was created

This section documents the creation and maintenance of this "read the docs" repository.

3.1 Prerequisites

- raspberry-pi
- github.com account and repository
- readthedocs.org account created/linked using your github login.

The "read the docs" theme for online documentation is used across the web for technical documentation and for lighter weight topics.

3.2 Raspberry pi set-up

It doesn't have to be a raspberry pi, but I've used one here.

First check to see that `git` is installed with `git --version`. Installed if need be:

```
sudo apt-get update
sudo apt-get install git
```

Now configure:

```
git config --global user.email "address@domain.com"
git config --global user.name "First Last"
```

and Install a couple of packages needed to build the docs:

```
sudo pip3 install sphinx
sudo pip3 install sphinx-rtd-theme
sudo pip3 install recommonmark
```

`sphinx` is a documentation generator that I've seen used in many places. This is a good "getting started" guide that I used:

- https://docs.readthedocs.io/en/stable/intro/getting-started-with-sphinx.html.
- I made some changes along the way, but it was a good place to start. Lots of trial and error.
- The `sphinx-rtd-theme` package is for the theme used for the documentation, and the `recommonmark` package enables the use of .md pages along with the .rst pages that are more commonly used.

In my home directory I keep all the github material in its own subdirectory created with `mkdir github`. Now I retrieve the information from github.com repo:

```
cd github
git clone https://github.com/essans/RasPi
```

Set some configurations:

```
cd RasPi

mkdir docs
cd docs

sphinx-quickstart
```

Running `sphinx-quickstart` enables quick set-up of some preliminary information which can be changed later. I then update the `conf.py` file which was auto-created during the quickstart. The most recent conf.py file can be found in the github repo in the `docs/source` folder.

3.3 Begin creating documentation

1. The "main" file is the `index.rst` file found in the `docs/source` folder. It contains text for the start of the docs and lists the other pages in the sequence in which they will be rendered. Create the other documents also as `.rst` files.
2. The `.rst` extension indicates "Restructured Text (ReST) formatting which is similar to markdown .md mark-up.
3. At first glance they both `.md` markup and ReST look similar in objectives but with different syntax but it seems that .md is see as a light-weight with `.rst` being favoured for use in technical documentation. Some discussion on this can be found here.
4. I've found a few references for looking up `.rst` syntax including:

- https://thomas-cokelaer.info/tutorials/sphinx/rest_syntax.html#introduction
- https://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html
- https://sphinx-rtd-theme.readthedocs.io/en/stable/demo/structure.html
- https://docutils.sourceforge.io/docs/user/rst/quickref.html

---

3.4 Build docs and push back to github

Once the `index.rst` and other pages are ready in the `/docs/source` folder i then `cd ..` up one level and build the documentation by running:

```
make clean
make html
```

There are usually some formatting errors that are flagged and will need to be fixed before running the above and only then proceed to...

Commit and push back to github.com with:

```
git add --all
git commit -m "an initial commit"
git push -u origin master
```

3.5 Import to readthedocs.org

Make any refinements to the docs via github editing and then when ready navigate to www.readthedocs.org, login, and go to the projects dashboard and click on "import a project" button. Select the repo that should be listed on the dashboard.

Once built the online docs are visible on https://raspi-recipes.readthedocs.io

The docs do not need to be re-built everytime a change is made as the updates can be made directly in `/docs/source/` aread of the the github repo.

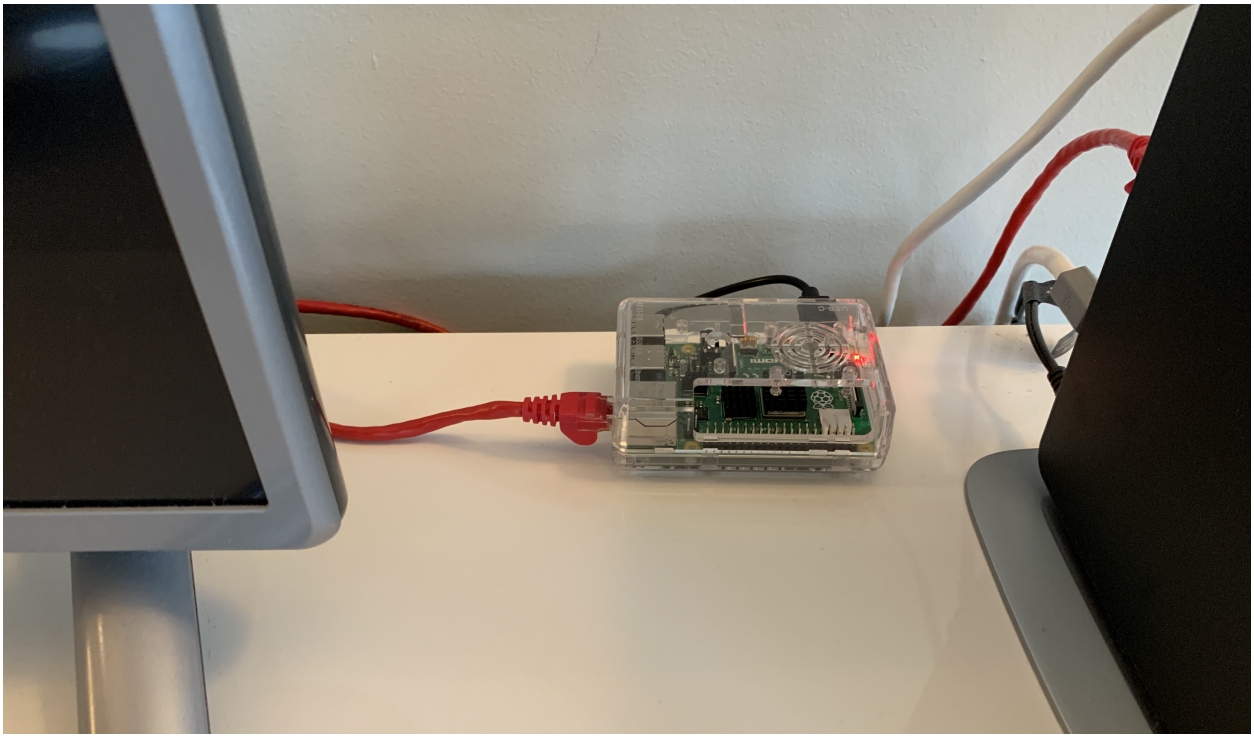
CHAPTER 4

Create wireless access point

4.1 Background

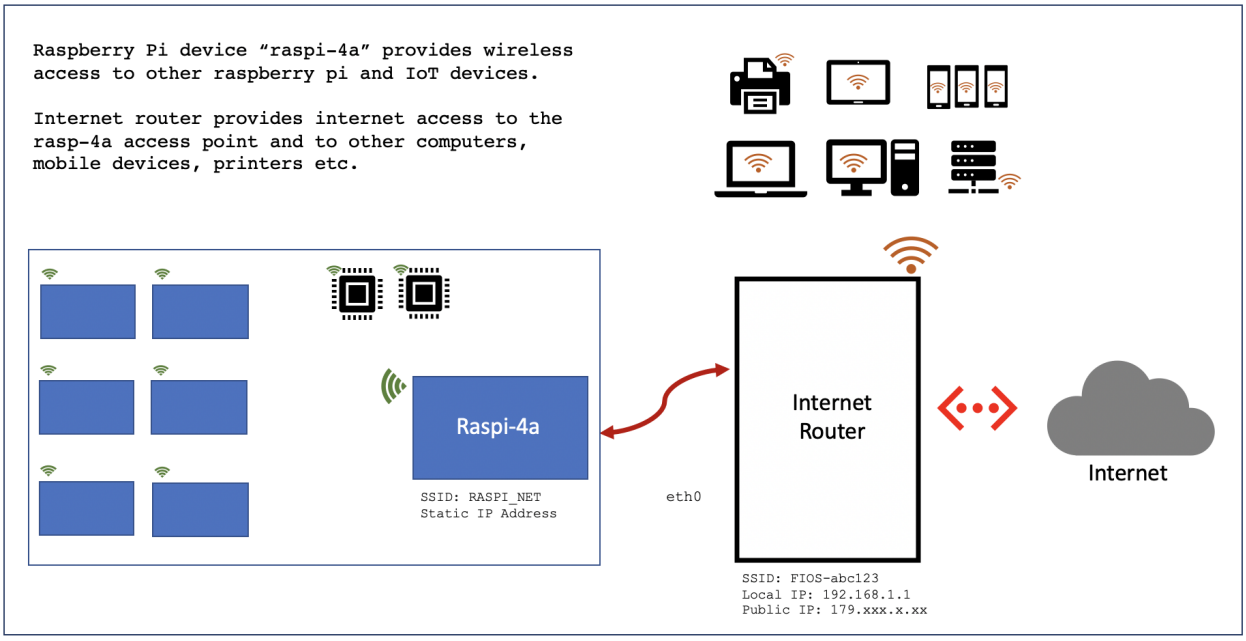
For the past few months I've had a Raspberry Pi 4 (raspi-4a) connected to my home network via ethernet cable and running 24/7. I can remote into the device using ssh from both within my local network and also from outside using port forwarding.

There are also other Raspberry Pi boards that connect to the same router via wifi. There are also computers, phones, iPads and other wireless devices that connect to the same wifi router. While every device has its own MAC address and operates on an assigned channel there is still a chance of conflicts, latency, and other non-optimal behavior when not using a router not designed to handle that capacity.



4.2 Objective

Reduce number of wifi devices connecting to the internet router by directing raspberry pi wireless traffic via the raspi-4a which is connected to the internet via ethernet. This raspi-4a device which acting as a wireless access point will broadcast its own SSID ("RAPI-NET"). The Raspi4a will still function normally as a raspberry pi while simultaneouly offering wireless access to any device with correct credentials.



4.3 Access Point set-up

This involves changes to various network configurations so it's a good idea to clone the SD Card so that there is a point to revert any changes back to. Also, changes in network settings may result in lost SSH connection so use a monitor and keyboard to be on the safe side...

(1) Check for updates

```
sudo apt-get update
sudo apt-get upgrade
```

(2) Install required packages

```
sudo apt-get install hostapd   # to create a wireless hotspot
sudo apt-get install dnsmasq   # easy-to-use DHCP and DNS server
sudo apt install bridge-utils  # to enable bridge between eth0 and wireless
```

(3) Switch off servces before changing any configurations:

```
sudo systemctl stop hostapd
sudo systemctl stop dnsmasq
```

(4) Edit the dhcpcd configuration file

```
sudo nano /etc/dhcpcd.conf
```

and add:

```
# These first 2 lines were added later after lots of trial and error...
# Both are needed to ensure that the bridge works correctly.
# They stop the eth0 and wlan0 ports being allocated IP
# addresses by the DHCP client on the Raspberry Pi

denyinterfaces wlan0
denyinterfaces eth0


# Next configure a static IP for the wlan0 interface

interface wlan0
static ip_address=192.168.4.1/24
nohook wpa_supplicant


# static IP address to enable ssh and also accessing
# the internet from the raspberry pi.
# This bit was also discovered after some trial and error...

interface br0
static ip_address=192.168.1.184/24  # assigning to AP
static routers=192.168.1.1
static domain_name_servers=8.8.8.8
```

(5) Restart the dhcp service:

```
sudo service dhcpcd restart
```

(6) Configure the DHCP server/masq configuration file

```
sudo nano /etc/dnsmasq.conf
```

and add:

```
interface=wlan0
dhcp-range=192.168.4.2,192.168.4.20,255.255.255.0,24h     # addresses for clients
```

The way to undersand this is that for `wlano` we are going to provide IP addresses between `192.168.4.2 and 192.168.4.20`, with a lease time of 24 hours. If providing DHCP services for other network devices (e.g. `eth0`), we would add more sections with the appropriate interface header, with the range of addresses intended to provide to the additional interface.

There are many more options for dnsmasq. See dnsmasq documentation for more details.

(7) Restart service

```
sudo systemctl start dnsmasq
```

(8) Configure the access point host software

and add:

```
nterface=wlan0
#driver=nl80211
bridge=br0
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
ssid=RASPI-NET  # choose ssid as desired
wpa_passphrase=<password_goes_here>

# hw_mode options above:
# a = a = IEEE 802.11a (5 GHz)
# b = IEEE 802.11b (2.4 GHz)
# g = IEEE 802.11g (2.4 GHz)
# ad = IEEE 802.11ad (60 GHz) (Not available on the Raspberry Pi)
```

The commented out `driver=nl80211` would have been needed if using as stand-one access point without a a bridge.

(9) Edit the following file:

```
sudo nano /etc/default/hostapd
```

to indicate location of the config file:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

(10) Enable and start service

```
sudo systemctl unmask hostapd
sudo systemctl enable hostapd
sudo systemctl start hostapd
```

and check status:

```
sudo systemctl status hostapd
sudo systemctl status dnsmasq
```

(11) Add routing and masquerade by first…

```
sudo nano /etc/sysctl.conf
```

and uncomment/enable this list:

```
net.ipv4.ip_forward=1
```

**(12) and then, add a masquerade for outbound traffic on eth0:

```
sudo iptables -t nat -A  POSTROUTING -o eth0 -j MASQUERADE
```

and save the iptables rule

```
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

(13) Edit the following file

```
sudo nano /etc/rc.local
```

and add the following line just above the "exit 0" so that the these rules install on boot:

```
iptables-restore < /etc/iptables.ipv4.nat
```

(14) Now, reboot the raspberry pi and the test before moving on to the next part of the set-up.

Using a wireless device, search for networks.

The network SSID specified in the hostapd configuration should discoverable, and it should be accessible with the specified password.

If SSH is enabled on the Raspberry Pi access point, it should be possible to connect to it with `ssh pi@192.168.4.1`

(15) Create bridge in order to share internet access

```
sudo systemctl stop hostapd

sudo brctl addbr br0    #add the bridge

sudo brctl addif br0 eth0    #make the connection
```

Create a file in order to create a linux bridge `br0` and add a physical interface `eth0` to the bridge:

```
sudo nano /etc/systemd/network/bridge-br0.netdev
```

and add these lines:

```
[NetDev]
Name=br0
Kind=bridge
```

(16) Configure the bridge interface br0 and the slave interface etho using .network files as follows:

```
sudo nano /etc/systemd/network/bridge-br0-slave.network
```

and add:

```
[Match]
 Name=eth0

 [Network]
 Bridge=br0
```

```
sudo nano /etc/systemd/network/bridge-br0.network
```

and add:

```
Match]
Name=br0

[Network]
Address=192.168.10.100/24
Gateway=192.168.10.1
DNS=8.8.8.8
```

then restart service:

```
sudo systemctl restart systemd-networkd
```

Use `brctl` to verify that bridge `br0` has been created.

Then reboot and run:

```
sudo systemctl unmask hostapd
sudo systemctl enable hostapd
sudo systemctl start hostapd
```

There should now be a functioning bridge between the wireless LAN and the Ethernet connection on the Raspberry Pi, and any device associated with the Raspberry Pi access point will act as if it is connected to the access point's wired Ethernet. The bridge will have been allocated an IP address via the wired Ethernet's DHCP server. Do a quick check of the network interfaces configuration via `ip addr`

4.3.1 References

The steps above resulted from much trial-and-error. Along the way the good use was made of the following articles with much insight gained:

https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md

https://thepi.io/how-to-use-your-raspberry-pi-as-a-wireless-access-point/

https://seravo.fi/2014/create-wireless-access-point-hostapd

https://howtoraspberrypi.com/create-a-wi-fi-hotspot-in-less-than-10-minutes-with-pi-raspberry/

http://raspberrypihq.com/how-to-turn-a-raspberry-pi-into-a-wifi-router/

https://www.instructables.com/id/Use-Raspberry-Pi-3-As-Router/

This might be interesting to explore one day. . . .

https://imti.co/iot-wifi/

CHAPTER 5

Cluster Computing

5.1 Overview

This recipe documets the steps taken in setting-up a cluster of Raspberry Pis. The idea is to try a few different approaches and in doing so reinforce understanding of some of the principles relating to linux networking, distributed computing and a few other things…

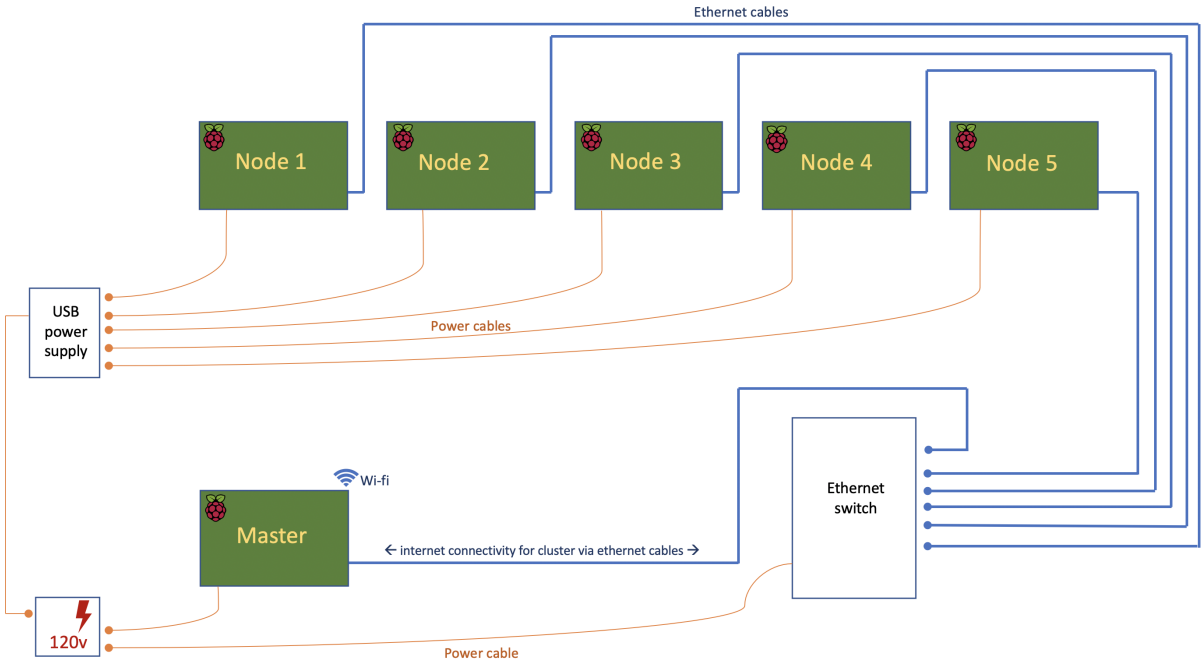
5.2 Architecture/Framework

The architecture for this first cluster will comprise of 5 Raspberry Pi 3 Model B+ worker nodes, powered by a USB power supply unit and networked together via a simple unmanaged switch. The master node will be a Raspberry Pi 4 (4gb RAM) on a separate power-supply wired into the same network switch.

The master node will connect to the internet via wireless LAN and using "IP Masquerading" will provide gateway access to the internet for the worker nodes only when needed.

Operating system on master and worker nodes will be Raspberry Pi OS flavor of the Debian distribution linux (www.raspbian.org).

Schematic



Parts List

Quantity	Part	Source	Cost
1	Raspberry Pi 4 (4gb)	Canakit	$ 55
5	Raspberry Pi 3 Model B+	Canaki	$ 175
5	Cat5e 1ft Ethernet patch cable	Amazon	$ 8
1	Cat6 18in Ethernet patch cable	Amazon	$ 5
1	GeauxRobot 6 layer dog bone stack	Amazon	$ 32
1	Clear Case for Raspberry Pi 4	Amazon	$ 5
1	Anker 60W PowerPort 6-port wall charger (2.4A per port max)	Amazon	$ 30
6	Sabrent 22AWG 1ft micro USB to USB cables	Amazon	$ 8
1	15W (5v, 3A) power supply (Raspi 4)	Pi Shop	$ 8
1	NETGEAR 8-Port Gigabit Ethernet Unmanaged Switch	Amazon	$ 22
1	Powerstrip with independent switches	Amazon	$ 20
6	SanDisk 32GB MicroSD HC ultra 80mb/s (non A1)	Amazon	$ 36
			$ 404

CHAPTER 6

Set-up of Master Node

This is the Raspberry pi which will control and manage the set of worker nodes.

6.1 Install and configure operating system

(1) See: https://raspi-recipes.readthedocs.io/en/latest/initialSetup.html

(2) Install python (in case not already installed), along with the fabric package that will be needed later.

```
sudo apt install python3-pip

sudo pip3 install fabric
```

(3) Install other tools that we'll need

6.2 Configure internet access for cluster

The Master Node will be the sole device on the cluster that connects to the internet. When worker nodes require internet access they will connect via the Master Node (if/when allowed). The set-up here is based on what was learned when configuring another Raspberry Pi to provide service as a secondary access point .

(1) Install linux command line utility dnsmasq and then stop the service before making configuration changes

```
sudo apt-get install dnsmasq

sudo systemctl stop dnsmasq
```

ref: https://en.wikipedia.org/wiki/Dnsmasq

(2) Edit the DHCP client daemon configuration file

```
sudo nano /etc/dhcpcd.conf
```

. . . adding the following lines at the bottom in order to assign a static IP address to the master node:

```
interface eth0
static ip_address=192.168.5.1/24
```

Save, exit, and then restart the service:

```
sudo service dhcpcd restart
```

(3) Control assignment of IP addresses to the worker nodes:

```
sudo nano /etc/dnsmasq.conf
```

After making sure that *every* line is commented out (usually the case, but there might be two at the bottom) add the following lines:

```
interface=eth0 # internet service to the nodes via ethernet
dhcp-range=192.168.5.2,192.168.5.64,255.255.255.0,24h # range of IP addresses
```

save, exit and then restart the service:

```
sudo systemctl start dnsmasq
```

(4) Enable IP forwarding:

```
sudo nano /etc/sysctl.conf
```

uncomment/enable this line:

(5) Now, iptables needs to be configured for ip packet filter rules

This is needed in order to allow all worker nodes to use the IP address of the master node when connecting to the internet. This is known as *masquerading* and the firewall keeps track of the incoming and outgoing connections (ie how to directly traffic to/from the relevant node) using Network Address Translation (NAT). Essentially by keeping track of ports and MAC addresses.

```
sudo iptables -t nat -A  POSTROUTING -o wlan0 -j MASQUERADE
```

and then save the rules so they are not lost upon reboot:

```
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

Then edit this file so that rules are installed upon boot:

```
sudo nano /etc/rc.local
```

and add the following line just above the "exit 0":

```
iptables-restore < /etc/iptables.ipv4.nat
```

Now reboot the master node. To list the rules in iptables:

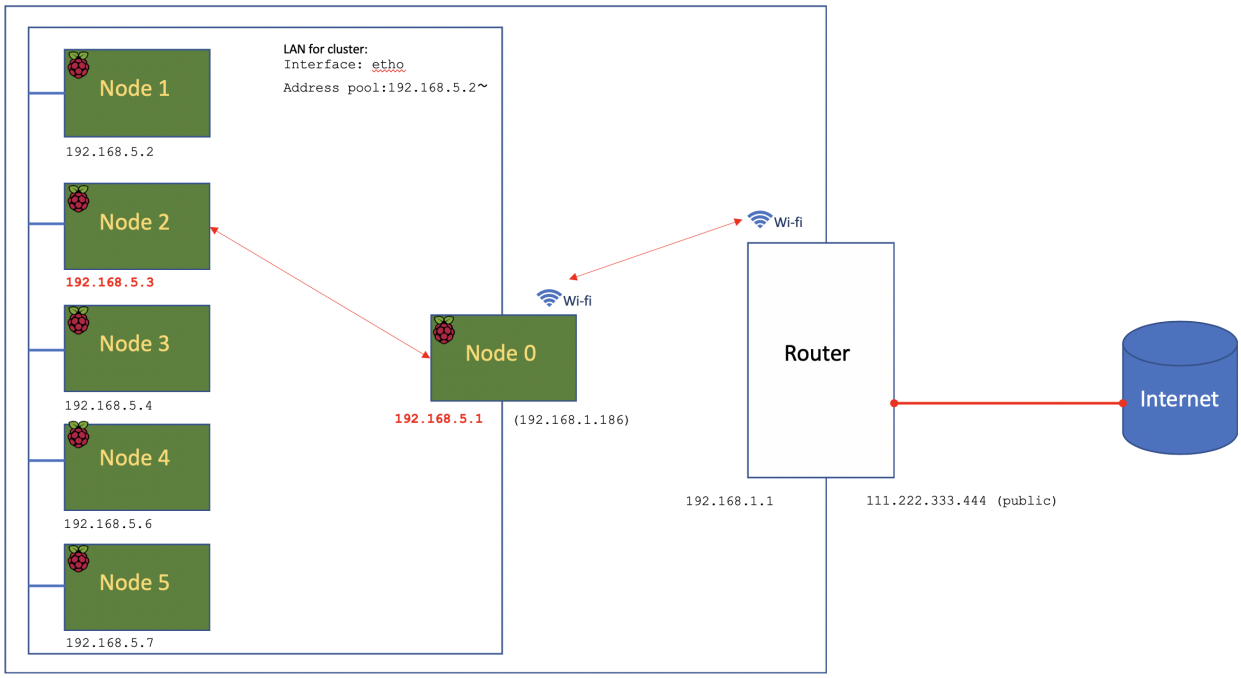
```
sudo iptables -t nat -L
```

To view connected devices:

```
arp -n
```

6.2.1 Overview

The following diagram illustrates how *masquerading* and network address translation will work once all nodes are set-up:



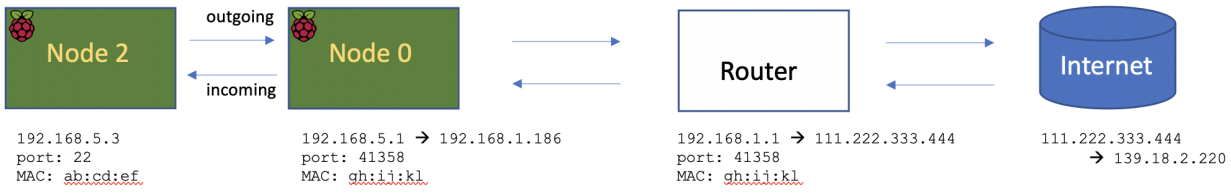
The way it works is as follows:

(1) When the worker nodes 1-5 come on line they will request an IP address from the DHCP server running on the master node. This will either be a new one, or the previously assigned one if available. At this point the IP address for each node is mapped to its corresponding MAC address.

(2) If node 2 seeks to connect to the internet (eg via a ping request sent via TCP on port 22) then that will travel to the master node. The master node using the DNS Masquerading will mask node2's IP address with it's own which will then travel to the router before itself betting masked with the router's public IP address.

At each step of the way mappings and tables are maintained so that when a response is received from the internet it knows how to find its way back to node2 which sits in an isolated part of the network.

```
Ping 139.18.2.220 -c 1
```



Node 2 can communicate outside of the cluster but nothing outside the isolated network can communicate in.

This can be seen in action using `tcpdump`

```
sudo tcpdump -i eth0 -en
```

6.2.2 Manage Internet Gateway

Much of the time access to the internet wont be required and hence the internet gateway can be disabled and then re-enabled whenever needed.

First locate the gateway while logged into the master node

```
sudo /sbin/route -n
```

The gateway will have a destination of 0.0.0.0 (or default if -n tail not used. To disable 192.168.1.1 gateway:

```
sudo /sbin/route del default gw 192.168.1.1
```

. . . and to add it back:

```
sudo /sbin/route add default gw 192.168.1.1
```

A handy script found here enables this to be automated:

```
GW="$(sudo /sbin/route -n | awk '$1=="0.0.0.0" {print $2; exit}')"
sudo /sbin/route del default gw "$GW"
echo "$GW" >~/my_tmp_file
```

the gateway is saved in a temp file enabling it to be re-enabled via a script

```
sudo /sbin/route add default gw "$(cat ~/my_tmp_file)"
```

—

The master node is now ready. It might make sense to back-up.

CHAPTER 7

Set-up of Worker Nodes

Connect components:

- Run power from power-hub to each worker node
- Connect each worker-node to network switch
- Connect master-node to internet switch

Switch on power to:

- Network switch
- Master-node
- Worker-node power-hub

7.1 Discover IP addresses for each worker node

In setting-up the master-node it was assigned a static ip address `192.168.5.1` with subnet mask of `255.255.255.0`. This can be confirmed with `ifconfig`

(1) Scan network

On the master-node scan that range using CIDR notation. For example `192.168.5.0/24` scans the 256 IP addresses from `192.168.5.0 to 192.168.5.255`. And, `192.168.5.0/16` scans all 65,534 IP addresses from `192.168.0.0 to 192.168.255.255`. The first scan takes a few seconds while the latter would take 30-60minutes...

```
sudo nmap -sn 192.168.5.0/24
```

This returns a list of IP addresses active on our isolated network.

(2) Identify each worker node in stack

Not absolutely necessary (indeed it would be impractical at scale and for remote servers), but we can from the master-node ssh into each pi and execute the following to "flash" the green LED to identify in the stack:

```
sudo sh -c "echo 1 >/sys/class/leds/led0/brightness"
```

By convention I assign node number from the top beginning with 1. Note them down the configuration looks like this:

```
192.168.5.1 Master aka raspi-4b (also reachable via 192.168.1.186 on wlan0)

192.168.5.41
192.168.5.42
192.168.5.19
192.168.5.8
192.168.5.9
```

7.2 Configure each worker node

In order to update/upgrade the OS on each worker node and make various configuration changes, we could take a number of approaches:

– ssh into each node and make these changes one at time. Just about manageable task for 5 nodes, but what if we had 50 nodes?

– Make all changes on one of the nodes and then clone the card for each of the other 4 nodes. Again a manageable task for 1+4 nodes, but what if I had 1+49 ? What if the worker nodes are not in the same physical location?

– The approach I take here is by using the `fabric` python package which allows programatic scheduling and running of shell commands over ssh. I can write some code that stores the IP addresses, user names, passwords etc for each node; loop across each of these node while passing the desired command lines we want to run. More information on the `fabric` python package can be found here: https://www.fabfile.org and here: https://docs.fabfile.org/en/2.5/

7.2.1 Manage worker nodes programmatically using fabric python package

On the master-node create the `~/code/python` folder, and then create a `cluster_exec_serial.py` file and copy/paste code from here: https://github.com/essans/RasPi/blob/master/Clusters/cluster_exec_serial.py

Create a `myconfigs.py` file in the same folder and copy the configs from here: https://github.com/essans/RasPi/blob/master/Clusters/myconfigs.py and update the IP addresses, passwords etc. Then run a `chmod u+x` to enable quick running from command line and and then run:

```
./cluster_config.py --help

# returns
#
# execute command across cluster

# optional arguments:
# -h, --help            show this help message and exit
# -p, --password        use passwords from config file instead of ssh keys
# -l, --logging         log only instead of printing commands to screen
# -s, --silent          do not show output (default: show output)
# -c COMMAND, --command COMMAND
                        command to execute (default: 'hostname -I')
```

(continues on next page)

(continued from previous page)

```
# -m, --master          include execution on master node

# -n [NODES [NODES ...]], --nodes [NODES [NODES ...]]
                        node numbers (default: 99 for all)
```

Test first using the following which should flash the green LED across each node including the master-node:

```
./cluster_config.py -p -c 'sudo sh -c "echo 1 >/sys/class/leds/led0/brightness"' -m Y
```

7.2.2 Update/Upgrade OS

Run an update/upgrade across all worker nodes, and reboot

```
./cluster_config.py -p -c 'sudo apt-get -y update'

./cluster_config.py -p -c 'sudo apt-get -y upgrade'

./cluster_config.py -p -c 'sudo shutdown -r now'
```

7.2.3 update localizations

Check, then update

```
./cluster_config.py -p -c 'timedatectl'
```

Raspberry Pi boards usually ship with the UK localization so we'll need to update if we're based in New York and the master is configured as such. The following will list available timezones: `timedatectl list-timezones`. And then to update:

```
./cluster_config -p -c 'sudo timedatectl set-timezone America/New_York'

./cluster_config.py -p -c 'timedatectl'  # to confirm updates
```

7.2.4 Update locale settings

Check, then update.

```
./cluster_config.py -p -c 'locale'
```

If updates are needed then first check that the locale is available:

```
./cluster_config.py -p -c 'locale -a'
```

If not then generate as needed: In this case for en_US first uncomment that line in the locale.gen file if necessary.

```
./cluster_config.py -p -c 'sudo sed -i "/en_US.UTF-8/s/^#[[:space:]]//g" /etc/locale.
↪gen' -n 1

# removes '# '
# to recomment a line with a trailing space:
# sed -i '/<pattern>/s/^/# /g' file


./cluster_config.py -p -c 'sudo locale-gen'

./cluster_config.py -p -c 'sudo update-locale LANG=en_US.UTF-8'

./cluster_config.py -p -c 'locale'  # to confirm
```

7.2.5 Change passwords

```
.cluster_config.py --p c 'echo -e "raspberry\nNewPassword\nNewPassword" | passwd'

# where NewPassword is the desired new password
```

Now update the passwords in the `myconfigs.py` script

7.2.6 Change hostnames

Update `hostname` for each pi from the "raspberrypi" default to "node1", "node2" etc. I could do these one at a time on each node via `raspi-config` or by updating these files:

```
/etc/hosts
/etc/hostname
```

..but instead I'll attempt this is one shot across all worker nodes remotely.

First I'll confirm the hostname of each node:

```
.cluster_config.py -p -c 'hostname -s'
```

These should all come back as "raspberrypi". In the above mentioned files I need to replace "raspberrypi" with "node1", "node2" etc. This could be done one at a time by passing the following as `-c` args to `./cluster_config.py`:

```
sed -i 's/raspberrypi/node1/g' /etc/hosts   #s to replace, /g global
sed -i 's/raspberrypi/node2/g' /etc/hosts
sed -i 's/raspberrypi/node3/g' /etc/hosts
sed -i 's/raspberrypi/node4/g' /etc/hosts
sed -i 's/raspberrypi/node5/g' /etc/hosts

# and then repeat for /etc/hostname
```

It's more interesting though to consider a "wrapper" script that calls `./cluster_config.py` in a loop:

```
#!/usr/bin/env python3

import sys
import subprocess


cmds_to_execute =   {1:"'sudo sed -i \"s/raspberrypi/node1/g\" /etc/hosts'",
                     2:"'sudo sed -i \"s/raspberrypi/node2/g\" /etc/hosts'",
                     3:"'sudo sed -i \"s/raspberrypi/node3/g\" /etc/hosts'",
                     4:"'sudo sed -i \"s/raspberrypi/node4/g\" /etc/hosts'",
                     5:"'sudo sed -i \"s/raspberrypi/node5/g\" /etc/hosts'"
                    }

for node,command in cmds_to_execute.items():

        cmd_to_send = "./cluster_config.py -p -c " + command + " -n " +str(node)

        subprocess.call(cmd_to_send, shell = True)
```

Above script is saved as `cluster_commands.py` and then run from the command line. Then re-run after updating the script with "/etc/hostname" instead of "/etc/hosts".

Lastly, reboot the worker nodes with `./cluster_config.py -p -c 'sudo shutdown -r now'` and confirm across the nodes that the hostnames have been updated.

7.2.7 Add all hostnames to each node

The `/etc/hosts` file needs to be further updated with ip addresses and corresponding hostnames for all nodes that form the cluster.

(1) First create and save a text file called `node` with the list of IP addresses and corresponding node IDs:

```
192.168.5.1    node0
192.168.5.41   node1
192.168.5.42   node2
192.168.5.19   node3
192.168.5.8    node4
192.168.5.9    node5
```

(2) Copy this file to the other nodes:

The following script `cluster_xfer.py` accepts arguments as described in the help and calls linux scp via a loop:

https://github.com/essans/RasPi/blob/master/Clusters/cluster_xfer_serial.py

But first create the required directories on each node:

```
./cluster_config.py -p -c 'mkdir code'
./cluster_config.py -p -c 'cd code && sudo mkdir python'

./cluster_config.py -p -c 'sudo chmod -R 0777 code'   #full permissions
```

Then copy the file across to each node, and then append the `node` file information to the `/etc/hosts` file:

```
./cluster_xfer_serial.py -p -f nodes -d '/home/pi/python'  #copy "node" file to all␣
↪nodes

cat nodes | sudo tee -a /etc/hosts  #update /etc/hosts file on master node

./cluster_config.py -p -c 'cd code/python && cat nodes | sudo tee -a /etc/hosts'
↪#same on workers
```

Then reboot everything

Now each node has the information required to reach other nodes. From any node (eg master) you can now ssh into another node (eg 2) with `ssh pi@node2`.

7.2.8 Create/copy ssh-keys

To simplify ssh access to the worker nodes from the master create public and private keys, and then copy the private keys to each worker.

```
cd ~/.ssh

ssh-keygen -t ed25519
```

When prompted leave the passphrase blank and set the name to id_cluster1

```
ssh-copy-id pi@node1
ssh-copy-id pi@node2
ssh-copy-id pi@node3
ssh-copy-id pi@node4
ssh-copy-id pi@node5

cat id_cluster1.pub >> authorized_keys    # needed if hdfs installed later
```

Now `ssh` into each node using the password and update various configurations by opening:

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

Uncomment/enable `PubkeyAuthentication yes` and enable `PasswordAuthentication no` and then reboot the node. The above operations can either be done one at a time or programmatically as shown earlier.

The usual way to `ssh` into each node (eg node1) would be to `ssh -i ~/.ssh/id_cluster1 pi@node1`. To simplify the process create a `config` file in the `~/.ssh` folder with the following entries and then save:

```
Host localhost
     User pi
     IdentityFile ~/.ssh/id_cluster1

Host node0
     User pi
     IdentityFile ~/.ssh/id_cluster1

Host node1
     User pi
     IdentityFile ~/.ssh/id_cluster1
```

(continues on next page)

(continued from previous page)

```
Host node2
     User pi
     IdentityFile ~/.ssh/id_cluster1

Host node3
     User pi
     IdentityFile ~/.ssh/id_cluster1

Host node4
     User pi
     IdentityFile ~/.ssh/id_cluster1

Host node5
     User pi
     IdentityFile ~/.ssh/id_cluster1
```

Now we can ssh into another node (say node1) using a simple `ssh node1`.

—

CHAPTER 8

Other Configurations

8.1 Disable red LEDs

As all nodes on the cluster will be accessed remotely and then shutdown remotely it is useful to confirm that everything has successfully powered down. Visually this is not possible as the red LED remains on even after the machine has been shut-down. But it is possible to disable the red LED while the Raspberry Pi is operational.

This blog post from Jeff Geerling shows how: https://www.jeffgeerling.com/blogs/jeff-geerling/controlling-pwr-act-leds-raspberry-pi

Execute across cluster:

```
./cluster_serial_exec.py -c 'echo 0 | sudo tee /sys/class/leds/led1/brightness' -m
```

LED's will remain disabled unless turned back on by placing `echo 1` in the above, or upon shutdown.

8.2 Add useful commands as aliases to `bashrc` file

```
alias ledoff="echo 0 | sudo tee /sys/class/leds/led1/brightness"
alias ledoff="echo 0 | sudo tee /sys/class/leds/led1/brightness"

alias clustercmd="python3 ~/code/python/cluster/clustercmd"
```

We can also update the bashrc file on other nodes:

```
clustercmd -c "echo ' ' >> ~/.bashrc"
clustercmd -c "echo '# my commands ' >> ~/.bashrc"
```

(continues on next page)

(continued from previous page)

```
clustercmd -c "echo 'alias ledoff=\"echo 0 | sudo tee /sys/class/leds/led1/brightness\
↪"' >> ~/.bashrc"

clustercmd -c "echo 'alias ledon=\"echo 1 | sudo tee /sys/class/leds/led1/brightness\"
↪' >> ~/.bashrc"

clustercmd -c "source ~/.bashrc"
```