**Captive Portal Web Art Machine**

So, you want to use your raspberry pi to serve your web application without any access to the internet? Excellent. Let's get on that. It will be easy. It will take a couple of linked blog posts to get this working, but is not more than about three hours of setup and watching things install, and when you're done, you will be able to turn on your computer and privately demonstrate your web app to whomever you like.

**Notes on Why You Would Even Care About This**

There are other ways of doing this. You could, possibly, serve your web app from your own phone. You could serve it from the magnificent Cloud, from your friend's hosting service, from Heroku, or from your laptop to the local environment, and I don't necessarily think this is *better* than doing that, except in the following instances.

1. You have no internet or external data services to speak of wherever you are going, such as the middle of a park, a desert, a forest, or a vault in the basement of an old building, or a catacomb.
2. You are in public and do not want to have to leave your own personal laptop lying around the site to be a server.
3. The internetless site you're attending is incredibly bad for delicate electronics, such as a desert made of the fine alkali powder known as gypsum.
4. You need this shit to be set up and left alone for, quite possibly, many hours with almost no maintenance.
5. The exhibit comes with an installation team and you don't want to have to be the installation team every day for the entire duration of the exhibit.
6. Your touch-enabled web game needs to be displayed in a bar, full of people with a great deal of booze and almost no sense. The bar's wiFi is already overloaded. It is *always* overloaded.
7. You want to be able to hide the server to get the "magic" feeling, and laptops are tough to hide.
8. Hidden things in small, waterproof boxes are not prone to death by metal fan's waterbottle landing directly on them in a concert environment.
9. Ruggedized computers are ludicrously expensive and considered an edge-case use (this is why OtterBox has a business).

These are all real situations I have actually had to contend with in the last three years, and I'm tired of them. Laptops are great for doing the work, but if what you need is something to run the application for people's own tiny pocketcomputers from a central locale, and you are going to have bandwidth troubles (you are going to have bandwidth troubles), what you want is a dedicated and reproducible set of hardware.

My day job is in IT for a major gallery that spends a lot of time and money trying to sort out how to safely exhibit various bits and bobs, and until we solve the problem of people who hate computers being in charge of exhibits, we won't have a coherent response to the Buying New Media For Galleries issue. This is profoundly insider baseball, but: your stuff isn't worth shit because we can't show it off because no-one knows how to maintain your new media bullshit but you, and everyone is pretty aware that in a competition between Rhizome and Degas, it's the Degas that's getting boots in the door. You'll have some legitimacy when you stop exhibiting awful processing sketches online as the direction of serious new art. Until then, it's down to the pirates at Art Basel, Marina Abramovic's expensively stuffy tents, and Damien Hirst, who has done a fantastic job of being the most representative possible artist of his demographic, the troll.

Showing things on the web is great, but building experiences in person is better, and being able to sell them to individuals is better yet.

Now let's make a computer that can serve our thesis code and rebel against the idea that you need to pay bandwidth fees to exhibit your art if it gets too popular.

**Projects This Project Has A Lot In Common With**

[subnod.es](http://subnod.es/tutorial.html), which sets up a chat server in a wiFi Pi, which is a private chat server to the local environment. Subnodes covers pretty much everything that the following tutorial also covers, although less in-depth. It's the same principal underlying both sets of work: privacy is important if you wish to speak freely.

Under that, though, I just don't care about chat clients, and I really *do* care about serving tasty full-on new media. The raspi is a really convenient platform for new media installations, because you can use one for every project. They're affordable. A lot of the time, "affordable" gets washed away in other considerations, but I find this to be a form of economic violence. It means that unaffordable gets to be the secret arbiter in the room, the thing which controls whether or not something works. So.

New media art is already quite violent in this respect, dependent as it is on the longest supply chains and the most interwoven manufacturing culture. I've written this on two computers, not including the pi, and done research with dozens more. I'm lucky. So I'm using some of that luck to make other people as lucky as I can.

As affordable as possible. Here we go.

**Things To Buy: About $125**

<https://www.sparkfun.com/> - Sparkfun is where you put money into the internet and get more internet back, this time in your house. You can buy most things listed here at Sparkfun. If you happen to live in Toronto, you should go to <http://www.creatroninc.com/> instead, because Creatron will give you an excellent deal.

<http://www.adafruit.com/products/1014> - The Raspberry Pi Learning Pack from Adafruit, which is like Sparkfun but with more black latex.

**Things To Own**

* A USB drive for putting your files on, because every operating system update will wipe your SD card.
* A spare television and an HDMI cable for that television, or an HDMI to DVI patch cable for your monitor. Ain't got either?
  + An ethernet cable and a network to your laptop that does not block your sharing internet through your laptop.
  + But really, this isn't a want, it is a need.
* A spare USB keyboard with extra USB holes for a mouse, which you will also want.

**Things to Know**

sudo is Linux for Do It Now.

apt-get is an inherited "package manager" from Debian linux. Your software needs other software. Debian, and therefore Raspi's base Wheezy distro, is good at maintaining these dependencies, largely because of apt-get. You are going to be hanging out with apt-get SO MUCH.

Things that follow sudo are commands. Some commands are not included in the raspian kit that you install when you format your SD card and install your operating system, because this is how Linux does not take fourteen hundred hours to install, unlike Windows.

So you will be getting very, very accustomed to typing

sudo apt-get install thisprogramname

and

sudo apt-get remove thisprogramname

which are how you install programs into your raspi.

These commands rely on the dark magic of what is called a Package Manager. Someone is doubtless making a dirty joke about that even now. If you must, you can think of a Package Manager as a phone book where the various programs used to build a Linux computer live until they are wanted. Yes, if there were no internet, this would be inconvenient, wouldn't it.

**Tutorials Chained In This Tutorial (it is tutorials all the way down in here)**

[Simon Monk on installing an OS to an SD Card from Adafruit](http://learn.adafruit.com/adafruit-raspberry-pi-lesson-1-preparing-and-sd-card-for-your-raspberry-pi/overview)

[Simon Monk on How To Wifi from Adafruit](http://learn.adafruit.com/adafruits-raspberry-pi-lesson-3-network-setup/setting-up-wifi-with-occidentalis)

[How To Prep Your Pi To Be A Webserver](http://www.dingleberrypi.com/2012/09/tutorial-prepare-your-raspberry-pi-to-become-a-web-server/)

[Josh On Design: Node on a Raspi in Five Minutes](http://joshondesign.com/2013/10/23/noderpi)

[How To Deploy NodeJS Applications, With Examples](http://gun.io/blog/tutorial-deploy-node-js-server-with-example/)

[Wifi and Ethernet from StackOverflow](http://raspberrypi.stackexchange.com/questions/8851/setting-up-wifi-and-ethernet)

[Subnod.es](http://subnod.es)

**Turning On Your Pi**

1. Image your microSD card using the [Simon Monk Tutorial](http://learn.adafruit.com/adafruit-raspberry-pi-lesson-1-preparing-and-sd-card-for-your-raspberry-pi/overview) on Imaging SD Cards.
2. Eject the microSD card and stick it into your RasPi.
3. Plug in your keyboard, and plug a mouse into your keyboard.
4. Plug in your HDMI cable and monitor. Turn them on.
5. Find the microUSB cable. This is your power cable. Clever. Stick it in.

Nothing happens.

Absolutely nothing whatsoever.

**Your Pi Case Is Almost Certainly Blocking Your Pi From Setup.**

Take it out of its case, put it on something static-free and carefully plug everything back in again. Plug in power *last.*

**Still Not Working.**

[Here is a fix from the forums](http://www.raspberrypi.org/phpBB3/viewtopic.php?f=28&t=47039&p=369594)

Take out the SD card, plug it into your working machine, and edit config.txt to uncomment the Safe Mode line.

hdmi\_safe=1 << this sucker right here.

This will let you see things even on your spare work monitor. It may not be the best idea long term. In the long term, you may also want to configure your pi to not use a graphic user interface, because you are not necessarily going to want to use your machine with a screen... directly.

**You can see an alarmingly lo-rez GUI.**

Authenticity score. Configure your Pi accordingly.

**Aside on Configuring a Pi to Be A Hotspot Through WiFi**

Here is an obvious thing, which did not occur to me when I started this project. A wiFi antenna can be used to receive information from the internet, or to broadcast its own hotspot, but it *cannot do both of those at the same time*. So you will eventually want to configure your pi to have both an ethernet plug for SSHing in to control the raspi, and a wifi antenna that talks to the internet to get package updates, which will eventually turn into a wifi antenna that serves a hotspot.

**Configuring Your WiFi To Get Things From The Internet**

Open up [How To Wifi from Adafruit](http://learn.adafruit.com/adafruits-raspberry-pi-lesson-3-network-setup/setting-up-wifi-with-occidentalis), which is quite simple.

Pro Linux Tip: Nothing will warn you if you mistype a folder name, say, adding an "s" to "network" to make it "networks." If you would like to confirm your folder name is correct, try typing "ls /etc/" to list the contents of that directory. Network is a default folder, and Interfaces is already present at first boot, so you can make sure your things are all there before you really get started.

The way to tell you have done something wrong is if you type the below command and an empty new file opens. You are *editing* a file here, not *creating*one.

From terminal

sudo nano /etc/network/interfaces

Then, in Nano, which is a text-editor (you could also use vi, in which case, get out of here you don't need me)

auto lo  
  
iface lo inet loopback  
iface eth0 inet dhcp  
  
allow-hotplug wlan0  
auto wlan0  
  
iface wlan0 inet dhcp  
wpa-ssid "network name, commonly called an ssid, goes here"  
wpa-psk "password"

then CTRL-X and Y to save it, then

sudo halt

Plug in your wifi antennae. It will turn on its little blue light if you did this correctly.

Everything should come up dandy.

If all went well above, you are now on your company wiFi.

Test this by entering:

sudo apt-get upgrade; sudo apt-get update

This will upgrade your raspbian to whatever the latest agreed-upon package lists are, then update those packages to their most recent approved version.

**Install NodeJS**

[This is pretty much from Josh on Design](http://joshondesign.com/2013/10/23/noderpi).

Create a directory for Node to live in:

sudo mkdir /opt/node

Get Node from your functional and well-tuned wiFi connection:

wget http://nodejs.org/dist/v0.10.2/node-v0.10.2-linux-arm-pi.tar.gz

Observe the cute ASCII progress bar.

Unzip (desticky from tarball) it:

tar xvzf node-v0.10.2-linux-arm-pi.tar.gz

Copy the contents of the newly unzipped folder and paste them to your new directory. This leaves a copy of the tar and a copy of the unzipped tar at their original locations. You can probably remove them using sudo rm when you're sure everything is where it should be.

sudo cp -r node-v0.10.2-linux-arm-pi/\* /opt/node

Edit - or create - a .bash\_profile file, which is a type of script that runs when you turn on the pi. In this case, it runs and tells Node that it exists on your computer, so that typing node runthisprogram will do something. [What *is*a .bash\_profile?](http://hacktux.com/bash/bashrc/bash_profile)

From your root directory:

sudo nano .bash\_profile to open a new nano text file

PATH=$PATH:/opt/node/bin

export PATH

Control-X, Y to save it.

Node lives in the /opt/node directory you created above. This adds the commands "node" and "npm" to what are called "environment variables," or "shit your computer understands when you type it at the terminal prompt." If you are curious, and god knows you must be to play with a raspi, you can type ls /opt/node/bin and see the little programs sitting there in their bin.

**Test Node Out, Make Sure It Can Install Modules**

[How To Globally Install Node Modules Even Though The Raspi Hates That](http://dougbtv.com/?p=302)

Now to test that Node can get some packages from the internet itself. This is important, because we would like the raspi to be responsible for reading and serving web apps to the internet, and the web apps to live on a USB key where they can be comfortably added, removed, backed up, and yanked when we don't want them there any more. Also lost in the wash, if you go for a very tiny key.

I did this by reading a great deal, and deciding I wanted to use forever to monitor and run screenPerfect, the weird HTML5 video app I've been working on. I also want to use Nodemon to automatically check if any of my files have changed, and to restart the server when they have.

The latter is, in this situation, a Nice To Have, not a Have To Have. I don't expect to push many updates through the internet to my internet-disabled installation box, but you know what would be fun?  
  
If I could do that. So why not throw in the kitchen sink.

To install a node package - or "module" - you type

npm install PACKAGENAME

To install one globally, type

npm install PACKAGENAME -g

To crush the thing under your boot and force it to install properly, follow the instructions in the link:

sudo su

PATH=/opt/node/bin/:$PATH

npm install PACKAGENAME -g

exit

That would be: become a super user, set where the super user account looks for the command npm, install the package globally, then exit the superuser privilege set.

You should go ahead and

npm install forever -g

npm install nodemon -g

[Per StackOverflow on running Nodemon and Forever together....](http://stackoverflow.com/questions/18803581/nodejs-nodemon-forever-give-me-an-error)

forever start /usr/local/bin/nodemon /path/to/YOURAPP.js

This should work to start and run your app, once you have your app, more or less forever. This is important, because your app will otherwise stream crash for no particular reason and you will not be able to work out how to fix it. Daemons are pretty great.

**Notes On Patience**

When I tried to install forever the first five times, it timed out, gave me a 404 error repeatedly, and declared I had insufficient permissions to do a global install no matter what. This is where computer science faith, confidence, and patience come in. When the install did not work for half an hour, I took a break, came back, and discovered that it installed the next day.

Try not to forget in this process that you're relying on a massive network of computers and people, and it's quite likely things are going to go wrong that are way out of your perception, much less control. Going for a break will help you keep patient and reduce the panic of typing what appears to be gibberish into a terminal all day long, in between crossing your fingers and sternly doubting the utility of your work.

Now! It is time to take away your keyboard, mouse, and monitor. You need to be able to use your second powered usb plug in order to load and unload your applications, and therefore, you need a new way to input information. Welcome to SSH.

**SSH via Direct Ethernet Connection and WiFi Internet Access**

[Raspi Stack Exchange: Ethernet and WiFi](http://raspberrypi.stackexchange.com/questions/8851/setting-up-wifi-and-ethernet)

In order to escape from the requirement of a keyboard and spare monitor, we have to turn on both your wifi and your ethernet at once. Although it appears to be Best Practice to use the wpa\_supplicant file to store how you wish your raspi to connect to the internet, I have had limited success with it, likely because I am not configuring a static IP for my raspi properly. I'm lazy, because I assume that at some point I will be passing the thing internet via my laptop, rather than via its own antennae.

My /etc/network/interfaces file looks like this:

auto lo

iface lo inet loopback

auto eth0

iface eth0 inet static

address [MY MAIN TERMINAL'S ETHERNET IP PLUS ONE]

auto wlan0

allow-hotplug wlan0

iface wlan0 inet dhcp

     wpa-ssid "network name here"

     wpa-psk "dubiously secure password"

I then followed the above tutorial regarding making your wiFi antenna stay on when you plug in the ethernet and start to use it.

sudo nano /etc/default/ifplugd

### MANY TALK, HOW COMMENT, SUCH WARNING ###

INTERFACES="eth0"

HOTPLUG\_INTERFACES="eth0"

ARGS="-q -f -u0 -d10 -w -I"

SUSPEND\_ACTION="stop"

This is an edit of the existing bits, and I can't tell if it will break everything long-term.

Here is, per the StackOverflow response above, what your startup script should read. This ensures that your wiFi antenna turns on, which is likely not something it was doing when you plugged in your ethernet directly.

sudo nano /etc/rc.local

#!/bin/sh -e

# Print the IP address

\_IP=$(hostname -I) || true

if [ "$\_IP" ]; then

  printf "My IP address is %s\n" "$\_IP"

fi

# Disable the ifplugd eth0

sudo ifplugd eth0 --kill

sudo ifup wlan0

exit 0

CTRL-X and Y to save, then reboot the thing and open a terminal on your main laptop. On your laptop, at the prompt, enter:

ssh pi@[the static ip address you entered under eth0 static above]

Your pi@[static ip] should appear in your terminal window, which means you can now talk to raspi. Per usual, to ensure your wifi is still working properly, try a sudo apt-get update. Or a ping google.com, both should return you data.

Okay! We have node installed, we have our ethernet turned on, we have our wiFi turned on, and every little blinky light is blinking its appropriate colour.

­­

**Captive Art Machine Part 2**In our previous chapter, [Part 1](http://irregularpattern.blogspot.com/2014/01/independent-art-machine-part-1.html), we addressed why you would want to do this, who is likely to want to make these machines happen, how to set up and turn on the wiFi to start setup, how to wire in your ethernet to SSH to your raspi so that you can do "headless" setup - ie, without a monitor or keyboard - and how to set up Node with some packages.

We also covered having faith and patience, because those are key elements to this type of work.

In this entry, we will be plugging in our USB and sorting out which parts of Node should live where, and perhaps a little about preparing Node apps specifically to live on a raspi installed inside a table/on a block of ice/in a thin coffin with a curious wire antennae. Just kidding. A different antennae will be a different post.

Unplug your things, put away your monitor, clear your keyboard to wherever it lives, and remember that you can at this point make an *image* of your SD card in case you need it to be reimaged suddenly in a later stage, such as when you turn your wiFi from being exclusively an internet eater into a hotspot provider.

**Mount Your USB Stick To Your Pi**

[How to Mount Your USB to Your Pi](http://raspi.tv/2012/mount-a-usb-flash-drive-on-raspberry-pi)

Once you manage this part, you will start to have *crazy* warnings on boot, because it's unlikely that all of these instructions are going down smoothly. This will make the Pi look for your *specific* memory card the moment it starts up. It may get cranky when the card is not present, such as the nth time you've removed it to plug in a keyboard and work directly with the tiny computer.

**Configuring Your Mount Drive**

This bears some thinking about, because the /media/ folder is clearly for media, and you are instead choosing to run a program off of the drive. subnod.es suggests making it your www drive, for world wide web, but subnod.es is also running everything off the SD card. Pick your poison! I picked /mnt/ because I plan to store my www apps on a USB stick, because what's better than having to chose between your programs and a keyboard.

You may choose differently! This bit sure is a pain in the ass.

For now, find your plugged-in USB gizmo by listing the the things plugged into dev:

sudo ls /dev/sd\*

If you've been following along, yours is almost certainly named "/dev/sda1".

So make a directory for it to be addressed at:

sudo mkdir /mnt/USBSTICKNAME;

Then mount it to that directory

sudo mount -t vfat -o uid=pi,gid=pi **/dev/sda1** /mnt/USBSTICKNAME/

Now you can poke around in your storage, which probably includes mainly your program.

sudo reboot

Which will kick you out of your Pi, so watch the lights on the darn thing until they're stable again, about two minutes, then:

ssh pi@[static ip]

Oh look. Your USB drive does not automatically mount at boot. Problem.

[How to Boot Mount External Memory](http://www.techjawab.com/2013/06/how-to-setup-mount-auto-mount-usb-hard.html)

Find out the actual name of your external memory card:

ls -l /dev/disk/by-uuid

Write down the UUID of your USB stick.

Take a moment to marvel that your normal operating system is clever enough to do this by default, wonder how it manages that. Detour through:

sudo apt-get install usbmount

and then return, because whatever, do it by hand per computer, why not.

sudo chmod 775 /mnt/USBSTICKNAME

sudo sp /etc/fstab /etc/fstab.bak

sudo nano /etc/fstab

Add:

UUID=YOURUUID /mnt/USBSTICKNAME vfat rw,defaults 0 0

CTRL-X, Y to save, then

sudo reboot

ls /mnt/USBSTICKNAME

... which should now display the contents of your USB key when you go looking for it.

The assumption here is that you're building a single, perfect arcade box to serve a single application in some very specific contexts. This is clearly not intended to serve as a mass distribution pi at this point, although working through this will perhaps get us to the point where we can simply share a Raspberry Pi image with our own art box software on it with the world, simple as imaging.

Wouldn't that be fun, a cartridge-based video game system for a tiny credit card sized pocket computer with exacting specifications that still needs to be mapped by hand every time you make one. Impractical! But not, maybe, more impractical than the original arcade machines, each board burned in forever.

Alternately, go through this entire tutorial, set everything up, read a copy of your SD card to your normal harddrive for backup, then copy the relevant applications to your SD to run. The risk here is running out of space for static files - a serious risk with video, especially large numbers of video games.

**Storing all your video game files and server files in the same place**

Although I have optimistically tried to make this a headless box, realistically, lots can go wrong with the SSHing process. You will probably eventually want a keyboard, and it is \_much\_ easier to store your access point as a single image per card, much like any other video game.

To store your games locally, rather than in the USB stick:

sudo cp -r /mnt/USBSTICKNAME /home/pi/YOURDIRECTORYNAME

Once this is working, it may be a good idea to back up your raspi to make sure you don't have to do this part again, as it can be time consuming.

**Now make sure you have everything you need from the internet.**

[Adafruit on Setting Up A WiFi Access Point](http://learn.adafruit.com/setting-up-a-raspberry-pi-as-a-wifi-access-point/connect)

This is the best guide to getting a wiFi access point set up on your pi. The key thing to remember is to download *everything you need* before you get started.

Skip halfway down the page to the Update Hostapd section and *do it first*. The instructions on getting you all WiFi'd up will destroy your ability to WiFi out to the internet, after all, so you need *all* your software in advance.

This is tricky, because isc-dhcp-server is a *son of a bitch* that simply does not run properly every time, for mysterious reasons of its own. It's good to go through that tutorial, but the very first thing we're going to to after it is complete is go sudo nano /etc/network/interfaces, comment out the bits about static wifi service, reboot, and

sudo apt-get autoremove isc-dhcp-server

sudo apt-get install hostapd dnsmasq

[Here is a super thorough how-to-configure DNSMasq tutorial.](http://www.the-hawkes.de/dnsmasq-a-local-dnsdhcp-server-on-raspberry-pi.html)

[Here is another one](http://www.iceflatline.com/2010/02/how-to-install-and-configure-dnsmasq/)

Configuration lives in /etc/dnsmasq.conf

Useful things included in those tutorials are: How to configure static hostname addresses, I assume so you can connect to specific "sites" when paired to the raspi. This will be super handy for you once you're trying to serve up a captive portal site at a specific address that is not listed in the global DNS registry.

There is also a bunch of theory that should be perceived as "detailed yet useful" because you probably want to route your visitors to DNS located *only* within your raspi, which is serving applications from static addresses to your local people.

Once configured, dnsmasq will just... run when you turn on your pi. You don't even have to log in.

My /etc/network/interfaces looks like this:

auto lo

iface lo inet loopback

auto eth0

iface eth0 inet static

     address 169.254.222.xx

allow hotplug wlan0

## wlan internet connect settings

#auto wlan0

#iface wlan0 inet dhcp

#     wpa-ssid "network name"

#     wpa-psk "network password"

iface wlan0 inet static

     address 192.168.42.1 #hahaha, 42

     netmask 255.255.255.0

I have it stored that way in order that I can readily switch between eating internet and serving something that looks like the internet, but is not, in fact, internetsy.

The trickiest part of this particular operation is the various IP addresses. IP addresses are raw internet magic. They are how we phone the internet, and how we talk to things once all the radios are tuned in, and they are admirable. From Wikipedia....

 Three ranges of IPv4 addresses for private networks were reserved in [RFC 1918](https://tools.ietf.org/html/rfc1918). These addresses are not routed on the Internet and thus their use need not be coordinated with an IP address registry.

Today, when needed, such private networks typically connect to the Internet through [network address translation](https://en.wikipedia.org/wiki/Network_address_translation) (NAT).

|  |  |  |  |
| --- | --- | --- | --- |
| **IANA-reserved private IPv4 network ranges** | | | |
|  | **Start** | **End** | **No. of addresses** |
| 24-bit block (/8 prefix, 1 × A) | 10.0.0.0 | 10.255.255.255 | 16777216 |
| 20-bit block (/12 prefix, 16 × B) | 172.16.0.0 | 172.31.255.255 | 1048576 |
| 16-bit block (/16 prefix, 256 × C) | 192.168.0.0 | 192.168.255.255 | 65536 |

Any user may use any of the reserved blocks. Typically, a network administrator will divide a block into [subnets](https://en.wikipedia.org/wiki/Subnetwork); for example, many [home routers](https://en.wikipedia.org/wiki/Residential_gateway) automatically use a default address range of 192.168.0.0 through 192.168.0.255 (192.168.0.0/24).

What this means is that you can basically name your pi what you like, but your home router default starts with 192.168.x.x, where your pi can start with just about anything. Subnod.es takes advantage of this by using the 10.x.x.x space - a Never Public space. You should use whichever of the above makes you happy, because we are not hosting even 200 people at once off a single Pi. This is what we call a "practical design constraint," which is where you think about what you want this thing to do and how much money you want to throw at it, and then build accordingly.

**Practical Design Constraint Note:**  
If you wish to take the long route, you can install [HAProxy](http://haproxy.1wt.eu/) and load-balance your Pi applications across more than one Pi. At this point, you are beyond my help, because you understand HAProxy. Why HAProxy and not ngnix like everyone else?  
  
ngnix does not play nice with websockets. Wah-wah.

**Once You're Connected and Your Pi is Serving Access...**

Your pi is now not particularly available on its ethernet connection. It's also not able to access external software itself! All of this is dire and inconvenient. You can ssh into the pi now if you're on its wiFi access point, so connect to that. Why?

We need to disconnect the keyboard and mouse and remount the USB stick, to make sure it's there at all.

If it's there at all, you can try running:

node /mnt/USBSTICK/yourappfolder/appname.js

And if that works, we're now ready for part 3, which is configuring the captive portal portion of the project.