#first update and upgrade the ubuntu instance

**sudo apt update && sudo apt upgrade**

# <https://docs.docker.com/engine/install/ubuntu/>

#Install docker

**sudo apt-get update**

**sudo apt-get install \**

**ca-certificates \**

**curl \**

**gnupg \**

**Lsb-release**

#you might need to install curl for this

**curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg**

#get keyring and repo

**echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null**

#update and install

**sudo apt-get update**

**sudo apt-get install docker-ce docker-ce-cli containerd.io**

#install nginx for reverse proxy

**sudo apt install nginx**

#install sqlite

**sudo apt install sqlite3**

#download the git repo for the server

**git clone https://github.com/WebThingsIO/registration\_server.git**

#now build the docker image (sqlite specific)

**sudo docker build –build-arg “db\_type=sqlite” -t registration-server . #CHANGE THE QUOTES, GOOGLE MESSES WITH ENCODING OF THEM AND WONT BUILD WITH DB\_TYPE**

#create directories that will serve both the configuration and data storage for the docker container

**mkdir /opt/docker/**

**chmod 777 /opt/docker**

**mkdir /opt/docker/registration-server**

**mkdir /opt/docker/registration-server/data**

**mkdir /opt/docker/registration-server/config**

#create a sqlite database in opt/docker/registration-server/data

**sqlite3 domains.sqlite**

#when the sqlite interface opens, run a blank query for the database to save

**sqlite> ;**

#now fix the configuration files to fit with our infrastructure

#here is what i have to get the thing up and running for now.

#pagekite.conf

<https://drive.google.com/file/d/1zkYv6-PKFX-y8Zl2-zzP6jkGUNSRLq7P/view?usp=sharing>

#pdns.conf

<https://drive.google.com/file/d/1fbEyMSpv3l3ydV5JrVGdGGXz13DkrMlZ/view?usp=sharing>

#config.toml

<https://drive.google.com/file/d/1lEvtEHGOlwo6dvSt2GcLjUOeh_fmh8b3/view?usp=sharing>

#these will all need to probably be changed before handing off to the client as it configures what our server will react to.

#place these files in the /opt/docker/registration-server/config directory

**mv /Path/to/file/config.toml /opt/docker/registration-server/config**

**mv /Path/to/file/pdns.conf /opt/docker/registration-server/config**

**mv /Path/to/file/pagekite.conf /opt/docker/registration-server/config**

#now we are almost ready to run our docker image

#however we might as well configure the nginx server first before we serve this up

#this config file goes into /etc/nginx/nginx.conf

**mv /path/to/nginx.conf /etc/nginx/**

#the nginx server needs this configuration changed on it for the proxy pass through

https://drive.google.com/file/d/1rCcn0rNEB34zshu8LFlv5pjKCbH86VNY/view?usp=sharing

#this config file serves the api.name.tld part up to the world (I think?)

#however we need to get rid of the default located in /etc/nginx/sites-enabled/default

**sudo rm /etc/nginx/sites-enabled/default**

#also we need ssl certificates from letsencrypt (work in progress cause we need a domain?)

#now we need to stop systemd-resolved and place it on a separate port so that our server can consume 53

#<https://medium.com/@niktrix/getting-rid-of-systemd-resolved-consuming-port-53-605f0234f32f>

**sudo systemctl stop systemd-resolved**

#now edit /etc/systemd/resolved.conf. This is described in the aforementioned link

**sudo nano /etc/systemd/resolved.conf**

#now we need to create a softlink to our new resolved.conf

**sudo ln -sf /run/systemd/resolve/resolv.conf /etc/resolv.conf**

#start systemd-resolved backup

**sudo systemctl start systemd-resolved**

#now run the docker image

**sudo docker run -d -v /opt/docker/registration-server/config:/home/user/config -v /opt/docker/registration-server/data:/home/user/data -p 127.0.0.1:81:81 -p 443:4443 -p 53:53 -p 53:53/udp --log-opt max-size=1m --log-opt max-file=10 --restart unless-stopped --name registration-server registration-server:latest**

#now this just needs to be configured on an amazon instance

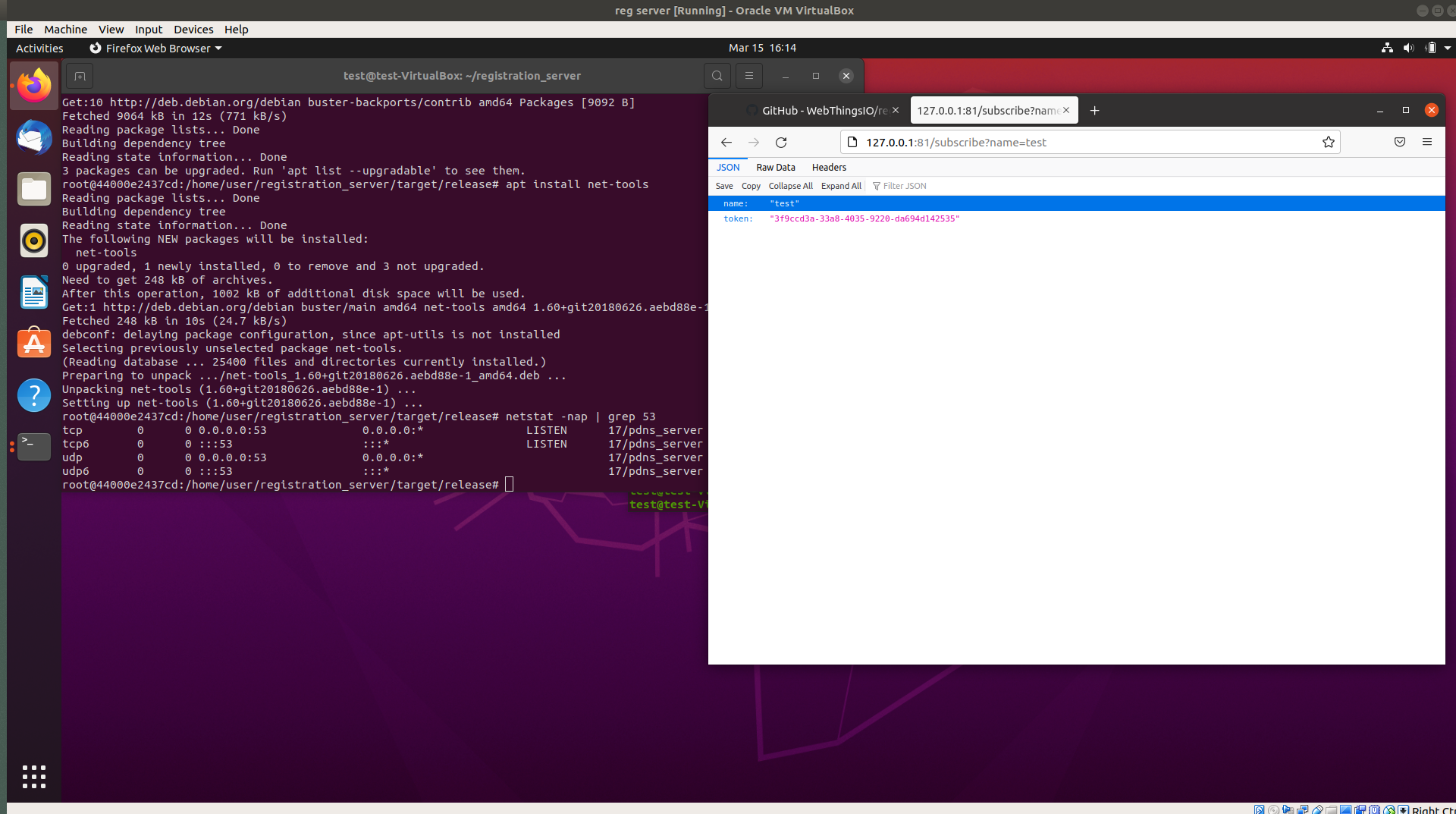
#to create an instance you just run through the wizard and select whichever instance you want

#please select an ubuntu 20.04 AMI

#amazon i believe can serve the hostname out using the route 53 service.

#once we have a domain name, Letsencrypt will finally be able to confirm that our server is available to the world and provide a certificate for our usage.

#also if you run into any issues with permissions (which you will) just chmod that thing with either 666 or 777



#I have a book as well im willing to pass out for the aws stuff.

#it has a section describing how/why to use route 53

<https://drive.google.com/file/d/1GhgUrc7yRHAyk0yxYWNuBi09XQefJRJI/view?usp=sharing>

#Check logging in docker

**docker logs -f registration-server**

#Certbot for the nginx server

# <https://certbot.eff.org/instructions?ws=nginx&os=ubuntufocal>

#these are the certbot commands to receive an automatic certificate from LE

#change the nginx conf to reflect the location of these certificates

**sudo snap install --classic certbot**

**sudo ln -s /snap/bin/certbot /usr/bin/certbot**

**sudo certbot certonly --nginx**

#as an aside, an A record must be created for the api.name.tld so there is some progress

#im currently trying different configurations on both route 53 and the registration server to see if i can get any further with this stupid stuff

#well it kinda works as it actually creates the domain on the registration server.

#this thing is real finicky and i have no clue what i fixed. But now it actually works

#hopefully it works in front of the client :D **Edit: It did not**

#also it doesnt send out emails. Which is what is probably breaking this thing. **Edit: who cares for emails**

#im looking into having aws serve as an email server/client **Edit: client may be able to set this up for us or something**

#i have now fixed this thing.

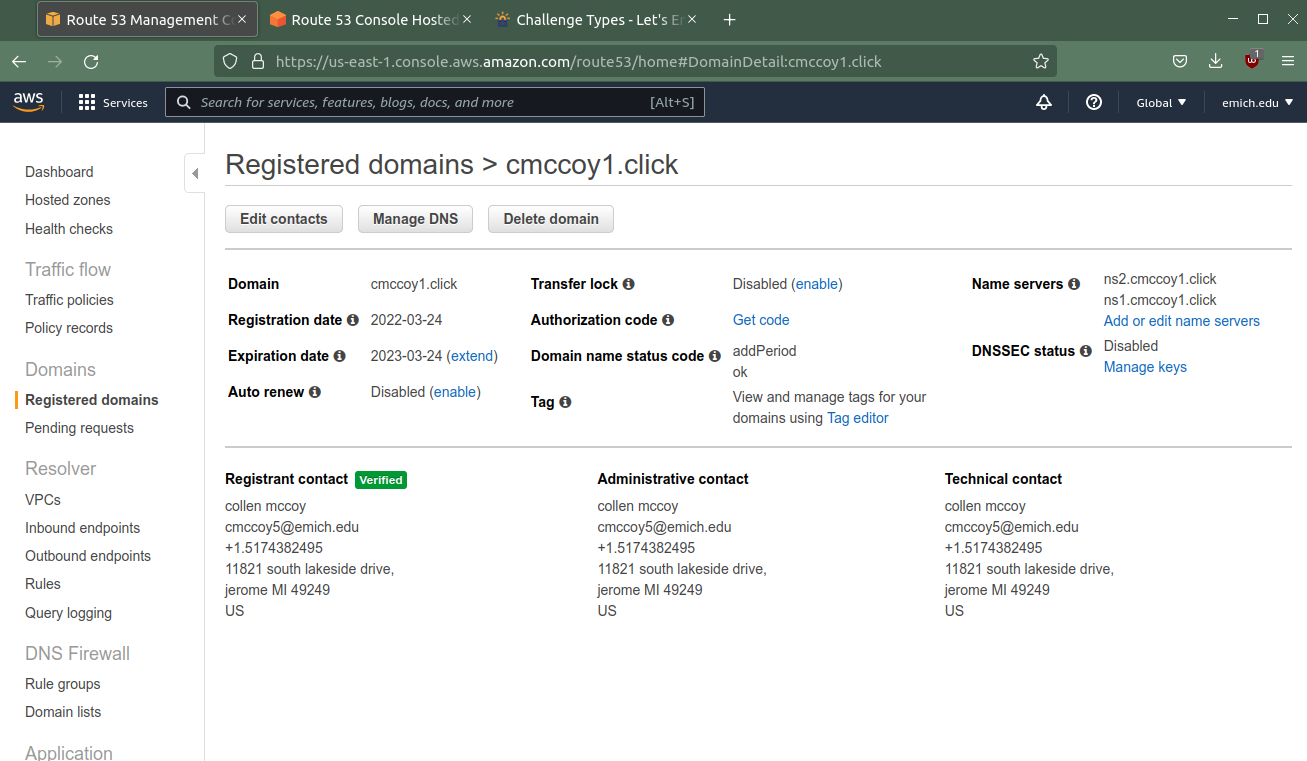
#after registration of the domain, you must remove the awsdns name servers

#notice how the [Name Servers] section only has two records (before it had 4 default awsdns servers and the ns1.cmccoy.click record)

#I believe what was happening prior was sometimes the server would use the other NS and sometimes it would use the ns1.cmccoy1.click server

#last thing to do is configure this thing securely.

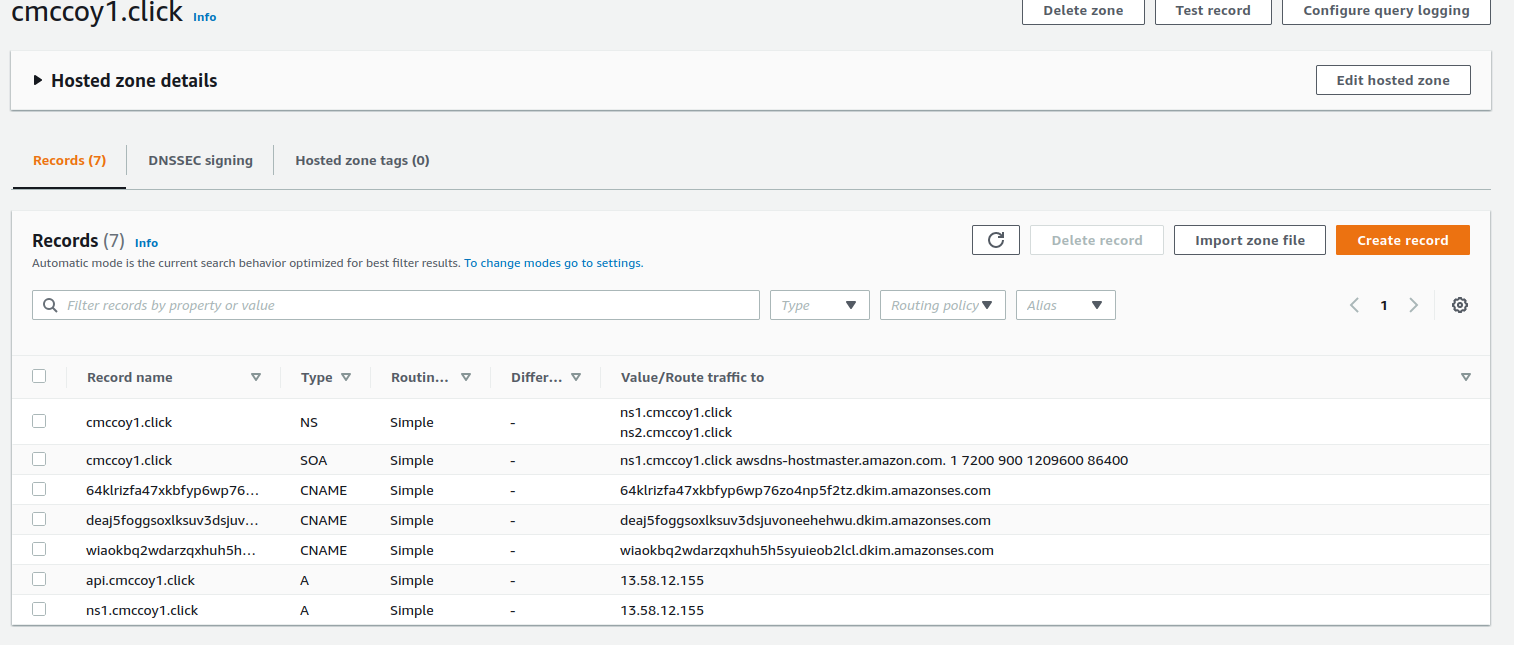
#securing this would mean just opening the ports needed to run and communicate with the server



#this is how your records should look

#i think you dont need an A record for the NS

#the cnames are for SES which is an added plus for your registration server



#Gateway Setup/Installation

<https://github.com/WebThingsIO/registration_server>

#make the gateway point to the reg server

#install the gateway

#cd into config and create a file named local.json

#add these directives to the json file

“””

{

"ssltunnel": {

"registration\_endpoint": "https://api.cmccoy1.click:8443",

"domain": "cmccoy1.click",

"certemail": "certificate@cmccoy1.click"

}

}

“””

#here is what you should get when you are successful

