

Docker DNS Server

Miraz Hossain | ID:181-15-10829

Project Submitted by:

MD. Miraz Hossain

ID: 181-15-10829

Section: A (O-1)

Department of Computer Science & Engineering

Project Submitted to:

Ms. Subhenur Latif

Assistant Professor

Faculty of Science & Information Technology

Daffodil International University

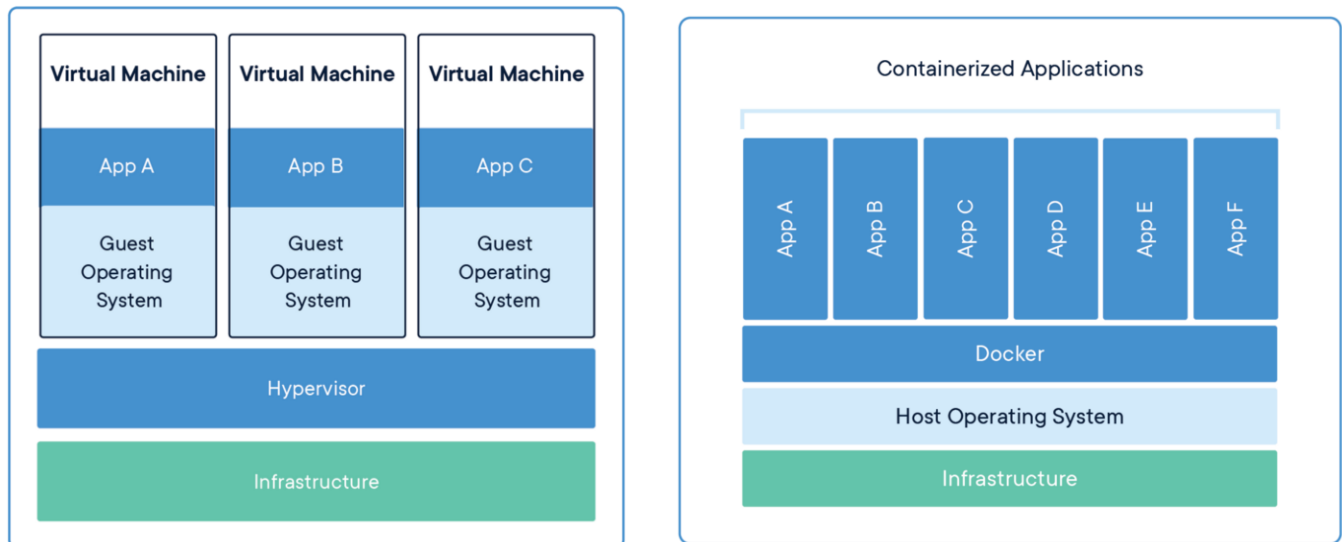
What I used to build Docker DNS Server:

- Ubuntu Server 20.04
- Docker Engine
- Pi-Hole Docker Image
- Flask with Python

Why had I used Docker?

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all the parts it needs, such as libraries and other dependencies, and deploy it as one package.

Docker is a bit like a virtual machine. But unlike a virtual machine, rather than



creating a whole virtual operating system, Docker allows applications to use.

the same Linux kernel as the system uses its core kernel and only requires applications be shipped with things not already running on the host computer. This gives a huge performance boost and reduces the size of the application.

And docker help a host server to run multiple application simultaneously. I have OpenVPN Access Server & DNS Server on a single host. In order to run both of them docker helps a lot. Both servers are isolated from each other by docker container.

Why I build a Docker DNS Server?

Pi-hole is a DNS server with powerful ads blocking service. Instead of browser plugins or other software on each computer, Pi-hole in one place and entire network is protected.

Network-level blocking allows to block ads in non-traditional places such as mobile apps and smart TVs, regardless of hardware or OS.

Voice Assistant Integration: Voice assistant like Alexa, Google Assistant, Cortana allows block & unblock certain websites like social networks, Entertainment Sites, CDN networks (block server from DNS level to prevent media, database as well as API) etc.

Server Configuration & Networking:

A DNS server cannot run on locally hosted VM to work networkwide such as router, smartphone, game console, TV etc. DNS server needs a public/external IP & always running so that any device or anyone can access it. Therefore, I used **Microsoft Azure** as cloud computing service.



Creating a virtual machine on azure:

Giving a resource group to my project:

Project details
Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

Deep Space Lab ▼

Resource group * ⓘ

(New) DNS-Server-RG ▼

[Create new](#)

Creating an Ubuntu Server 20.04 instance:

Instance details

Virtual machine name * ⓘ

DNS-Server ✓


Region * ⓘ

(US) East US ▼

Availability options ⓘ

No infrastructure redundancy required ▼

Image * ⓘ

 Ubuntu Server 20.04 LTS - Gen1 ▼

[See all images](#)

Azure Spot instance ⓘ

☐

Size * ⓘ

Standard_B1s - 1 vcpu, 1 GiB memory (\$7.59/month) ▼

[See all sizes](#)

Configuring Network Interface:

Giving a name to virtual network & creating a range of subnets for server:

Network interface

When creating a virtual machine, a network interface will be created for you.

Virtual network * ⓘ

(new) DNS-Server-RG-vnet

Create new

Subnet * ⓘ

(new) default (10.0.0.0/24)

Public IP ⓘ

(new) DNS-Server-ip

Create new

NIC network security group ⓘ

None

Basic

Advanced

Public inbound ports * ⓘ

None

Allow selected ports

Select inbound ports *

HTTP (80), HTTPS (443), SSH (22)

Opening port 53 for DNS, 80/443 for web interface, 943 for call web request:

DNS-Server | Networking

Virtual machine

»

Attach network interface

Detach network interface

dns-server911

IP configuration ⓘ

ipconfig1 (Primary)

Network Interface: dns-server911

Effective security rules

Troubleshoot VM connection issues

Topology

Virtual network/subnet: DNS-Server-RG-vnet/default

NIC Public IP: 13.72.79.79

NIC Private IP: 10.0.2.4

Accelerated networking: Disabled

Inbound port rules

Outbound port rules

Application security groups

Load balancing

Network security group DNS-Server-nsg (attached to network interface: dns-server911)

Impacts 0 subnets, 1 network interfaces

Add inbound port rule

Priority	Name	Port	Protocol	Source	Destination	Action	
300	SSH	22	TCP		Any	Allow	...
320	HTTP	80	TCP	Any	Any	Allow	...
340	HTTPS	443	TCP	Any	Any	Allow	...
350	DNS-Protocol	53	TCP	Any	Any	Allow	...
360	DNS-Protocol-UDP	53	UDP	Any	Any	Allow	...
370	Voice-Assistant-Protocol	943	TCP	Any	Any	Allow	...
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow	...
65001	AllowAzureLoadBalancerInBo...	Any	Any	AzureLoadBalancer	Any	Allow	...
65500	DenyAllInBound	Any	Any	Any	Any	Deny	...

Giving a public/external static IP for DNS & DNS name level for web interface:

DNS-Server-ip | Configuration ...

Public IP address

Search (Ctrl+/) << Save Discard Refresh

Overview

Activity log

Access control (IAM)

Tags

Settings

Configuration

Properties

Locks

Monitoring

Alerts

Metrics

Diagnostic settings

Logs

Automation

Tasks (preview)

Assignment

☐ Dynamic ☒ Static

IP address ⓘ

13.72.79.79

Idle timeout (minutes) ⓘ

4

DNS name label (optional) ⓘ

mirazdsl-dnsserver .eastus.cloudapp.azure.com

Alias record sets

Want to closely track this Public IP address? Create an alias record in Azure DNS. [Learn more.](#)

[+ Create alias record](#)

Subscription	DNS zone	Name	Type	TTL
No results.				

The server is now up & running and ready to install Docker Engine.

Now, logging in to server through PuTTY software using IP **13.72.79.79**

Prerequisite:

Ubuntu has a build-in local DNS resolver which can create conflict with pi-hole docker DNS resolver. To run pihole docker DNS without conflict local DNS resolver should be disabled. After that adding an external DNS such as 1.1.1.1 or 8.8.8.8 for local resolver through **“nano”** at **/etc/resolv.conf**

```
sudo systemctl stop systemd-resolved.service
```

```
sudo systemctl disable systemd-resolved.service
```

Installing Docker Engine on ubuntu server:

First, updating my existing list of packages:

```
sudo apt update
```

Next, installing a few prerequisite packages which let apt use packages over HTTPS:

```
sudo apt install apt-transport-https ca-certificates curl  
software-properties-common
```

Then adding the GPG key for the official Docker repository to my system:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg |  
sudo apt-key add -
```

Adding the Docker repository to APT sources:

```
sudo add-apt-repository "deb [arch=amd64]  
https://download.docker.com/linux/ubuntu focal stable"
```

Next, updating the package database with the Docker packages from the newly added repository by **apt update**.

Finally, installing Docker Engine:

```
sudo apt install docker-ce
```

****Source & additional information of Docker is [here](#)**

A snippet of docker engine from my ubuntu server:

```
DNS Server
Selecting previously unselected package docker-ce-cli.
Preparing to unpack .../2-docker-ce-cli_5%3a20.10.5~3-0~ubuntu-focal_amd64.deb ...
Unpacking docker-ce-cli (5:20.10.5~3-0~ubuntu-focal) ...
Selecting previously unselected package docker-ce.
Preparing to unpack .../3-docker-ce_5%3a20.10.5~3-0~ubuntu-focal_amd64.deb ...
Unpacking docker-ce (5:20.10.5~3-0~ubuntu-focal) ...
Selecting previously unselected package docker-ce-rootless-extras.
Preparing to unpack .../4-docker-ce-rootless-extras_5%3a20.10.5~3-0~ubuntu-focal_amd64.deb ...
Unpacking docker-ce-rootless-extras (5:20.10.5~3-0~ubuntu-focal) ...
Selecting previously unselected package slirp4netns.
Preparing to unpack .../5-slirp4netns_0.4.3-1_amd64.deb ...
Unpacking slirp4netns (0.4.3-1) ...
Setting up slirp4netns (0.4.3-1) ...
Setting up containerd.io (1.4.4-1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/containerd.service → /lib/systemd/system/containerd.service.
Setting up docker-ce-cli (5:20.10.5~3-0~ubuntu-focal) ...
Setting up pigz (2.4-1) ...
Setting up docker-ce (5:20.10.5~3-0~ubuntu-focal) ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /lib/systemd/system/docker.socket.
Setting up docker-ce-rootless-extras (5:20.10.5~3-0~ubuntu-focal) ...
Processing triggers for man-db (2.9.1-1) ...
```

```
DNS Server
Setting up docker-ce-rootless-extras (5:20.10.5~3-0~ubuntu-focal) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for systemd (245.4-4ubuntu3.6) ...
miraz@DNS-Server:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2021-04-10 04:41:38 UTC; 34s ago
 TriggeredBy: ● docker.socket
    Docs: https://docs.docker.com
   Main PID: 3177 (dockerd)
     Tasks: 8
    Memory: 43.0M
    CGroup: /system.slice/docker.service
            └─3177 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Apr 10 04:41:37 DNS-Server dockerd[3177]: time="2021-04-10T04:41:37.232524252Z" level=warning m>
Apr 10 04:41:37 DNS-Server dockerd[3177]: time="2021-04-10T04:41:37.232622459Z" level=warning m>
Apr 10 04:41:37 DNS-Server dockerd[3177]: time="2021-04-10T04:41:37.232930782Z" level=info msg=>
Apr 10 04:41:37 DNS-Server dockerd[3177]: time="2021-04-10T04:41:37.453537050Z" level=info msg=>
Apr 10 04:41:37 DNS-Server dockerd[3177]: time="2021-04-10T04:41:37.578546695Z" level=info msg=>
Apr 10 04:41:38 DNS-Server dockerd[3177]: time="2021-04-10T04:41:38.347088962Z" level=warning m>
Apr 10 04:41:38 DNS-Server dockerd[3177]: time="2021-04-10T04:41:38.347661206Z" level=info msg=>
Apr 10 04:41:38 DNS-Server dockerd[3177]: time="2021-04-10T04:41:38.347933426Z" level=info msg=>
Apr 10 04:41:38 DNS-Server systemd[1]: Started Docker Application Container Engine.
Apr 10 04:41:38 DNS-Server dockerd[3177]: time="2021-04-10T04:41:38.434482965Z" level=info msg=>
lines 1-21/21 (END)
```

Note: every snippet can be view as high-resolution by clicking on them.

Pulling Pi-Hole image from Docker Hub:

Creating a shell script file using nano to pull pi-hole image on my docker server:

```
docker run -d \  
  --name pihole \  
  -p 53:53/tcp -p 53:53/udp \  
  -p 80:80 \  
  -p 443:443 \  
  -p 943:943 \  
  -e TZ="America/New_York" \  
  -v "$(pwd)/etc-pihole:/etc/pihole/" \  
  -v "$(pwd)/etc-dnsmasq.d:/etc/dnsmasq.d/" \  
  --dns=127.0.0.1 --dns=1.1.1.1 \  
  --restart=unless-stopped \  
  --hostname pi.hole \  
  -e VIRTUAL_HOST="pi.hole" \  
  -e PROXY_LOCATION="pi.hole" \  
  -e ServerIP="13.92.153.59" \  
  pihole/pihole:latest  
  
printf 'Starting up pihole container '  
for i in $(seq 1 20); do  
  if [ "$(docker inspect -f "{{.State.Health.Status}}" pihole)" == "healthy" ] ; then  
    printf ' OK'  
    echo -e "\n$(docker logs pihole 2> /dev/null | grep 'password:') for your pi-hole:  
https://${IP}/admin/"  
    exit 0  
  else  
    sleep 3  
    printf '.'  
  fi  
  
  if [ $i -eq 20 ] ; then  
    echo -e "\nTimed out waiting for Pi-hole start, consult container logs for more  
info (\`docker logs pihole\`)"  
    exit 1  
  fi  
done;
```

Raw code also can be found [here](#)

Next, giving bash command to shell script:

```
sudo bash pihole.sh
```

Now, the docker pi-hole image is pulling from docker hub:

```

: Docker-Engine x | +
deadadmin@Docker-Engine:~$ sudo bash pihole.sh
sudo: unable to resolve host Docker-Engine: Resource temporarily unavailable
WARNING: Localhost DNS setting (--dns=127.0.0.1) may fail in containers.
Unable to find image 'pihole/pihole:latest' locally
latest: Pulling from pihole/pihole
75cb2ebf3b3c: Extracting [=====>] 19.04MB/22.52MB
0db2707d1040: Download complete
a9232b8f3e23: Download complete
3a6d9813a3cf: Download complete
321d5430e809: Download complete
47b3363e9dca: Download complete
6f57a163a34c: Download complete
04bbc0b48ef4: Download complete
2579eafbc15c: Downloading [=====>] 84.36MB/110.1MB
018c74957032: Download complete
a64527f12a74: Download complete
fdc2ebbe8553: Download complete
11921923a10e: Download complete
█
```

The server is now ready.

```

: Docker-Engine x | +
deadadmin@Docker-Engine:~$ sudo bash pihole.sh
sudo: unable to resolve host Docker-Engine: Resource temporarily unavailable
WARNING: Localhost DNS setting (--dns=127.0.0.1) may fail in containers.
Unable to find image 'pihole/pihole:latest' locally
latest: Pulling from pihole/pihole
75cb2ebf3b3c: Pull complete
0db2707d1040: Pull complete
a9232b8f3e23: Pull complete
3a6d9813a3cf: Pull complete
321d5430e809: Pull complete
47b3363e9dca: Pull complete
6f57a163a34c: Pull complete
04bbc0b48ef4: Pull complete
2579eafbc15c: Pull complete
018c74957032: Pull complete
a64527f12a74: Pull complete
fdc2ebbe8553: Pull complete
11921923a10e: Pull complete
Digest: sha256:abdddfb266dd8e0591f97203ad11fd8dc33f2542187223f20ff18862b76bfb
Status: Downloaded newer image for pihole/pihole:latest
d768d683f78b8f4bd412f9c329adb55a34c7512ca4be6847396886e4dacdbbc5
Starting up pihole container ..... OK
Assigning random password: sLr1Uzo4
Setting password: sLr1Uzo4 for your pi-hole: https://admin/
deadadmin@Docker-Engine:~$
deadadmin@Docker-Engine:~$
deadadmin@Docker-Engine:~$
deadadmin@Docker-Engine:~$ █
```

Logging-in to pi-hole container:

Firstly, to log-in to pihole container I must use docker commands:

```
sudo docker exec -it pihole bash
```

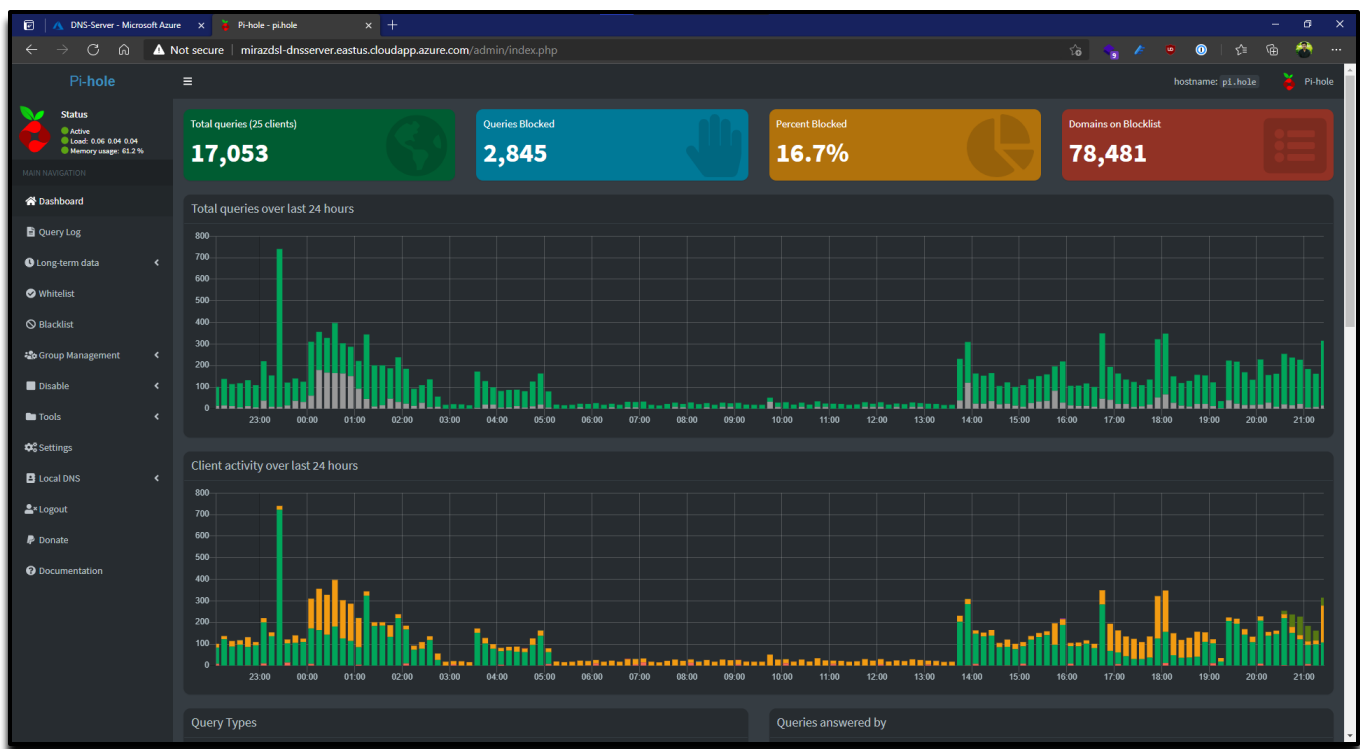
Now changing pi-hole container's random created password:

```
pihole -a -p
```

After that, accessing our DNS server from public internet:



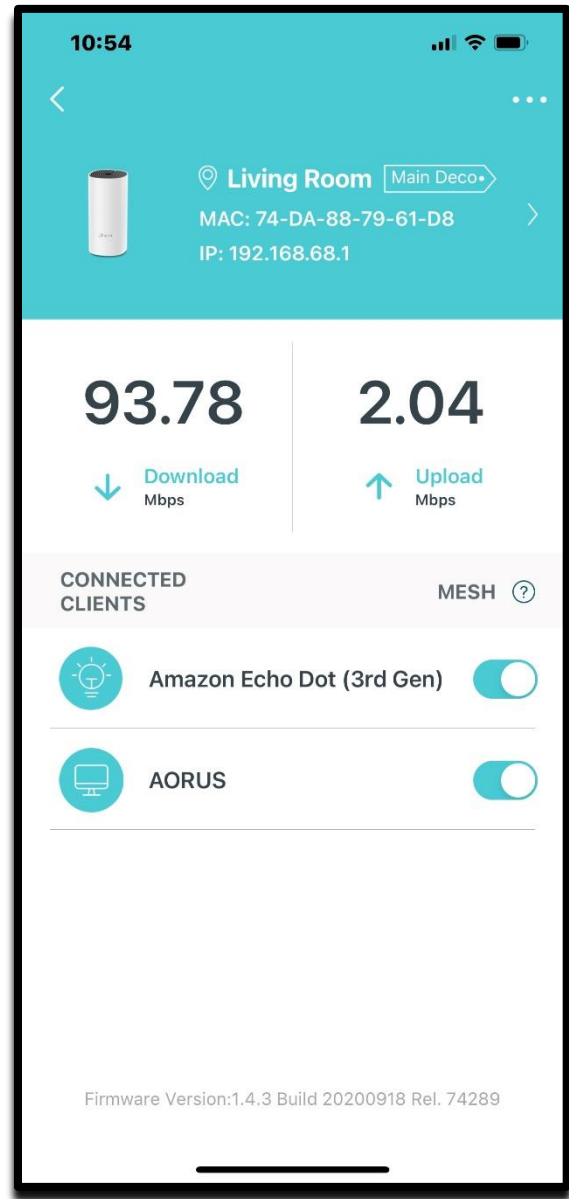
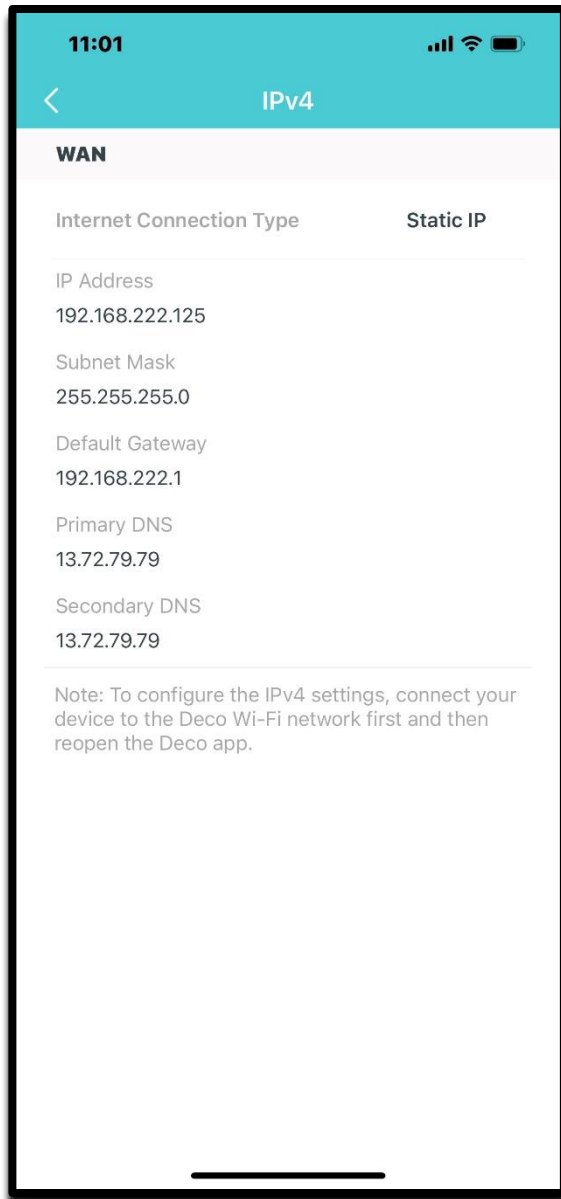
<http://mirazdsl-dnsserver.eastus.cloudapp.azure.com/admin>



**Source & additional information of Pi-hole is [here](#)

Configuring DNS on TP-Link router:

The address of my DNS server is: **13.72.79.79**



Now it is working fine & blocking annoying ads on websites.

Allowing DNS server to Interact with virtual assistant:

Using Flask Framework (python), I opened a port (943) on my public IP specially to get call-request:

```
import time
import os
from flask import Flask, render_template, request
app = Flask(__name__)
@app.route("/<deviceName>/")
def action(deviceName):
    if deviceName != 'monstermash':
        if deviceName == 'block':
            os.system('./blockregex.sh')
        if deviceName == 'unblock':
            os.system('./unblockregex.sh')

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=943)
```

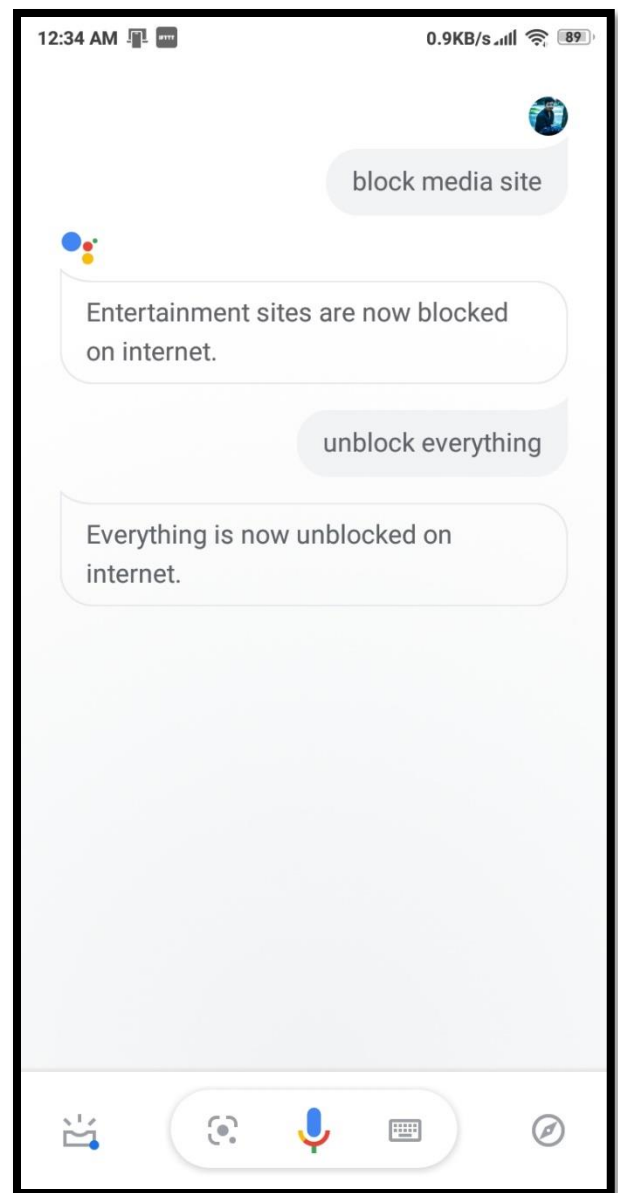
Raw code also can be found here: [show me](#)

When Flask get a call-request on **“13.72.79.79:943/block or /unblock”** it will execute a bash command on a specific shell file. And those files are coded with pi-hole regex filter.

- blockregex.sh - Raw code can be found on blob-storage [here](#)
- unblockregex.sh - Raw code can be found on blob-storage [here](#)

[Left] Configuring IFTTT to post a call-request from Google Assistant.

[Right] giving commands to google assistant to block certain media sites which is coded on blockregex.sh & unblockregex.sh





<https://github.com/Miraz4300/docker-dns-server>



<https://www.linkedin.com/in/miraz4300/>



server configuration & adding ad list to DNS server video from scratch:

<https://drive.google.com/drive/folders/1belifgjosouKesDP9rmG6lTgIN7hNoVr?usp=sharing>

****This project is made possible with Microsoft Azure, Ubuntu Server, Docker, Pi-hole, Python, Flask & IFTTT.**

THANKS

a u t h o r : m i r a z h o s s a i n