

**VACCINE MONITORING USING LOCALIZED BACKEND
COMMUNICATION WITH REST AND SELF -HOSTED API**

ABSTRACT

In real-world application, projects using public and open-source API will not suffice the needs of a private organization and its security is questionable. To overcome this, creating a local hosted API to Store data is more secure as it is not connected to the internet. Also gives us flexibility and reliability compared to an open-source environment.

TABLE OF CONTENTS

SERIAL NO.		TITLE	PAGE NO.
		ABSTRACT	2
		ACKNOWLEDGEMENT	3
1	1.1	INTRODUCTION	5
	1.2	OBJECTIVES AND GOALS	7
	1.3	APPLICATIONS/ FEATURES	8
3		DESIGN	
	3.1	BLOCK DIAGRAM	14
	3.2	SYSTEM ANALYSIS	
	3.3	IMPLEMENTATION STEPS WITH SCREENSHOTS	16
4		CODING AND ANALYSIS	
5	5.1	RESULT	
5	5.2	CONCLUSION AND FUTURE WORK	38
	5.3	COST ANALYSIS	39
		REFERENCES	40
6		BIODATA	42

1.1 INTRODUCTION

Web applications often follow a client-server model meaning that there is a piece of software which runs in your web browser (the client) and a piece of software which runs on a server somewhere.

Self-hosting is the activity of having and administrating your own server, typically at home, to host your personal data and services yourself instead of relying exclusively on third-parties. In typical web apps, someone else owns the server, stores your information, and runs the application.

Here's where self-hosting is different. Self-hosting is when you put an application on your server, store the information in your own database, and create your own web experience, that is, any information related from data to implementation such as: the hosting, the domain registration, the software and the maintenance and security can be handled over a single platform without any need of reliable external software to manage the server.

This also removes the limitations such as data size, user limit, cost and scalability of the server, in short there are no real limits. And if an error occurs you won't have to go to tech support or wait for hours to get a response as the user can modify as per their choice.

Basically, having your own self-hosted site allows you ultimate flexibility and customization. We completely own the content and the site and you can do whatever you want with it.

- Have more control over everything about your site
- Have more storage
- Totally free to use and implement

Decentralization is an additional approach we considered in self-hosting, it is a way of querying resources, except rather than the resources being hosted at a few large data centers controlled by Google, Amazon, etc. the files are decentralized and hosted across various nodes run by anyone across the world.

1.2 OBJECTIVES AND GOALS

- To create a self-hosted backend API that can be used by an infinitely scalable NodeMCU microcontroller to gather data in a structuralist text format in a localized setting.

1.3 APPLICATIONS/FEATURES

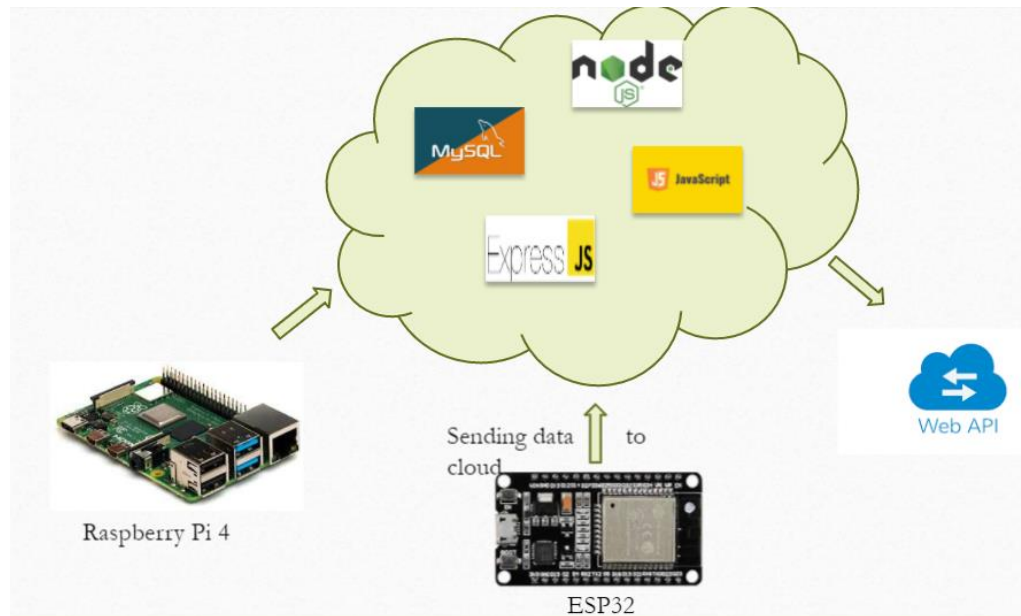
By self-hosting services and creating our backend API, we can decrease the long-term cost down to minimum hardware and power. With secure protocols and DDOS protection, the service can be internet-proofed. The documentation and features can be built individually allowing for ease of use or extreme specialization.

- **Flexibility** – Hardware can be decided by the server-side and can be scaled up dynamically as the demand increases along with traffic.
- **Privacy and Data Protection** – The data that is passed through the self-hosted server is completely transparent and can be encrypted securely without the scepticism that accompanies giant cloud service providers.
- **More control than hosted platforms can offer**- Due to the fact it's your server and framework that you're building, you have miles more control and flexibility of what actually appears in your store.
- **You're not limited customisation**-wise and can include anything you like on your site.
- **Switching platforms is easier**-As the platforms are usually low cost, if you need to switch having outgrown a supplier, you can do so a lot cheaper and easier than hosted platforms.
- Benefit of self-hosting is that the user has complete control over their data, at a potentially lower monthly cost.

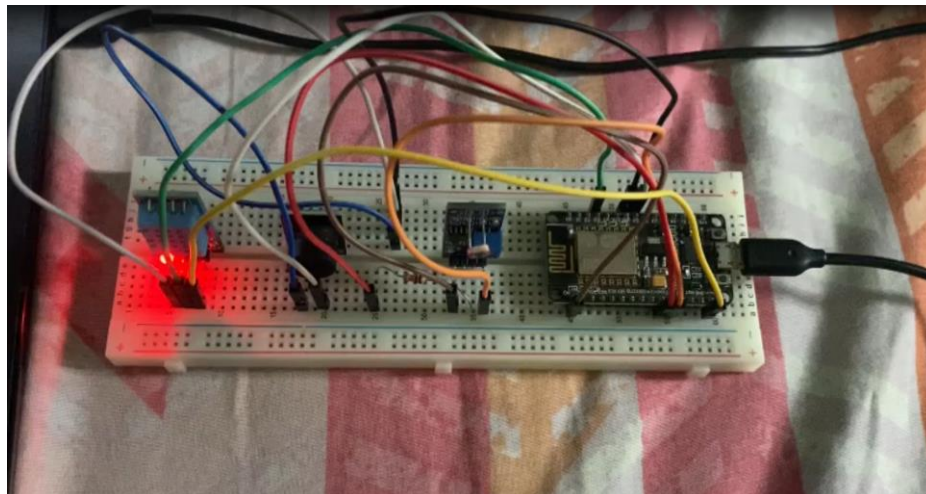
Based on many of the comparative advantages of cloud hosting mentioned above, many organizations and government agencies are migrating their Information systems to cloud hosting as it proves to be more cost-effective, flexible, secure and reliable in the long term. For example, banks can use self-hosted cloud servers for some of their applications despite the sensitiveness of their data with careful assessment of the pros and cons of cloud solutions, some organizations are encouraged to build a framework for firms and organizations that wish to move to the new approach and recognize the diversity of business models and services available”

3. DESIGN

3.1 BLOCK DIAGRAM



3.2 CIRCUIT



3.2 SYSTEM ANALYSIS

3.2.1 HARDWARE USED:

Raspberry Pi 4(Hosting Device)

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python.

ESP32

ESP32 can perform as a complete standalone system or as a slave device to a host MCU, reducing communication stack overhead on the main application processor. It can interface with other systems to provide Wi-Fi and Bluetooth functionality through its SPI / SDIO or I2C / UART interfaces.

3.2.2 SOFTWARE USED:

Docker

Docker is an open-source project that makes it easy to create containers and container-based apps. Originally built for Linux, Docker now runs on Windows and MacOS as well.

Dockerfile

A Dockerfile is a text file written in an easy-to-understand syntax that includes the instructions to build a Docker image. A Dockerfile specifies the operating system that will underlie the container, along with the languages, environmental variables, file locations, network ports, and other components it needs—and about the container once we run it. Each Docker container starts with a Dockerfile.

Container:

Containers isolate applications' execution environments from one another, but share the underlying OS kernel. They're typically measured in megabytes, use far fewer resources than VMs, and start up almost immediately.

Portainer

It is an open-source tool for managing containerized applications. It works with Kubernetes, Docker, Docker Swarm, Azure ACI in both data centers and at the edge. It removes the complexity associated with orchestrators so anyone can manage containers.

It can be used to deploy and manage applications, observe the behavior of containers and provide the security and governance necessary to deploy containers widely. Here, it is a lightweight management UI which allows you to easily manage your Docker host or Swarm cluster. It is meant to be as simple to deploy as it is to use. It consists of a single container that can run on any Docker engine (Docker for Linux and Docker for Windows are supported).

It allows you to manage your Docker stacks, containers, images, volumes, networks, etc. It is compatible with the standalone Docker engine and with Docker Swarm.

NGINX

NGINX is open-source software for web serving, reverse proxying, caching, load balancing, media streaming, etc. It started out as a web server designed for maximum performance and stability. With Nginx one master process can control multiple worker processes. The master maintains the worker processes, while the workers do the actual processing.

Since Nginx is asynchronous, each request can be executed by the worker concurrently without blocking other requests.

Some common features seen in Nginx include:

Reverse proxy with caching, IPv6, Load balancing, FastCGI support with caching, WebSockets, Handling of static files, index files, and auto-indexing and TLS/SSL with SNI.

Node.js

- Node.js is an asynchronous event-driven JavaScript runtime, which is designed to build scalable network applications.
- Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.
- Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser.
- Consequently, Node.js represents a "JavaScript everywhere" paradigm, which unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts.

3.3 IMPLEMENTATION STEPS WITH SCREENSHOTS

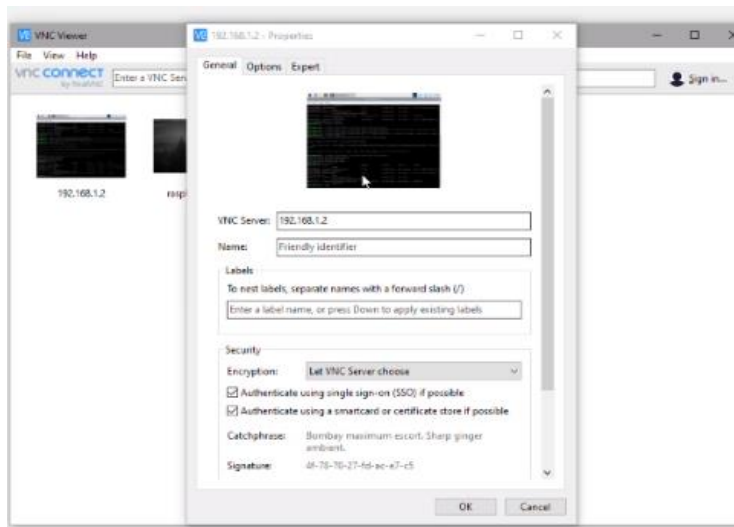
Step 1: Getting started

First is to get a device for implementing our platform which can handle all the load and process without any interference.

The best choice we found was RPI 4 ARM 64 architecture, a mini computer used for this project that is capable of handling heavy load sporting a faster 1.5GHz clock speed processor installed with the latest Raspbian OS in it.

Step 2: Creating the main interface -

- Backend - Server side
- Front end - Client Side



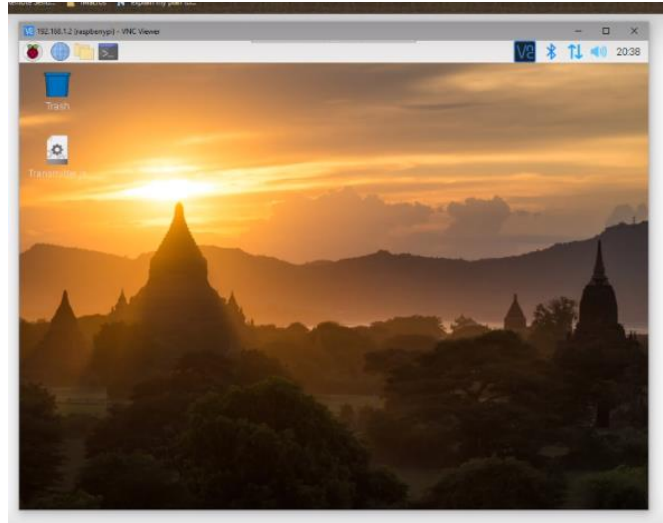
To setup an ip address for making a connection between our system and rpi

Here we use VNC to act as an interface between the rpi4 and our PC which helps to remotely control the desktop interface of one computer (running VNC Server) from another computer

We need to configure a default ip address so that all information from data storage to user service is handled from a single platform for better management and monitoring the server task. Also making it process faster with less effort with no interference of other applications to create problems while running.

Step 3: Finishing the setup and interface for installing necessary software

Once all setup of the device is done with the creation of a remote interface. Now we need an application which can act as client service to create a hosting platform within the website and a web server which can perform the task given with high capacity, stability and low resource consumption. We chose mainly two software for deploying the server: Docker and Nginx



Docker - to create the server platform which uses OS-level virtualization to deliver software in packages called containers which enables you to separate your applications from your infrastructure so you can deliver software quickly.

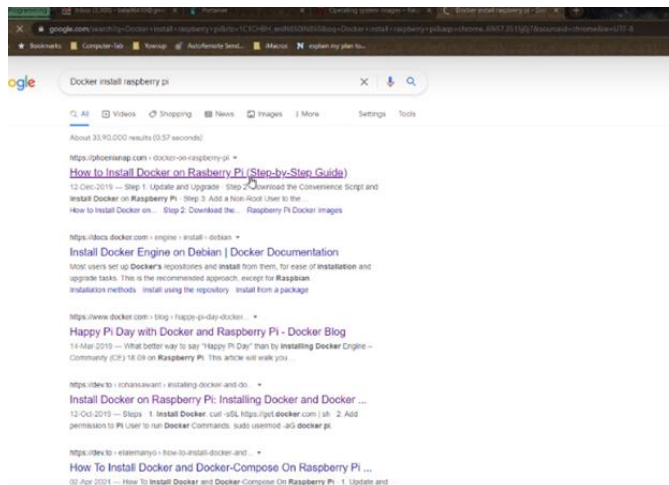
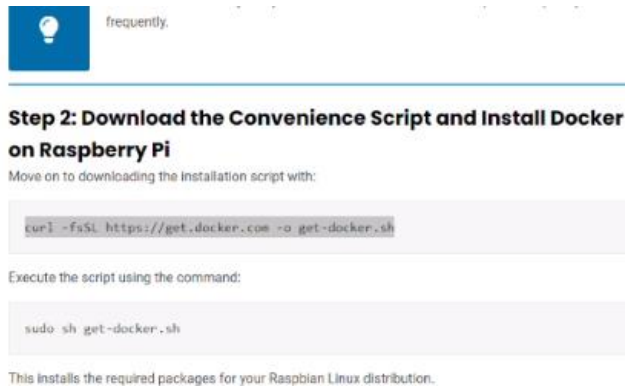


Figure 3.2.3 This shows where we can download the Docker for raspberry pi



To run docker on your pi, we use this script to load all commands needed to create an allocated VM port to deploy an individual host client.

Running the script to load necessary resources for docker

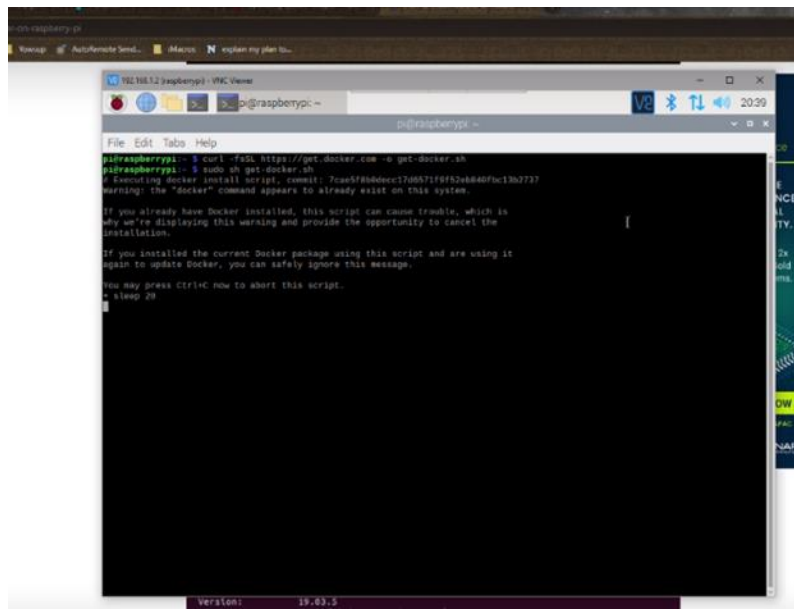
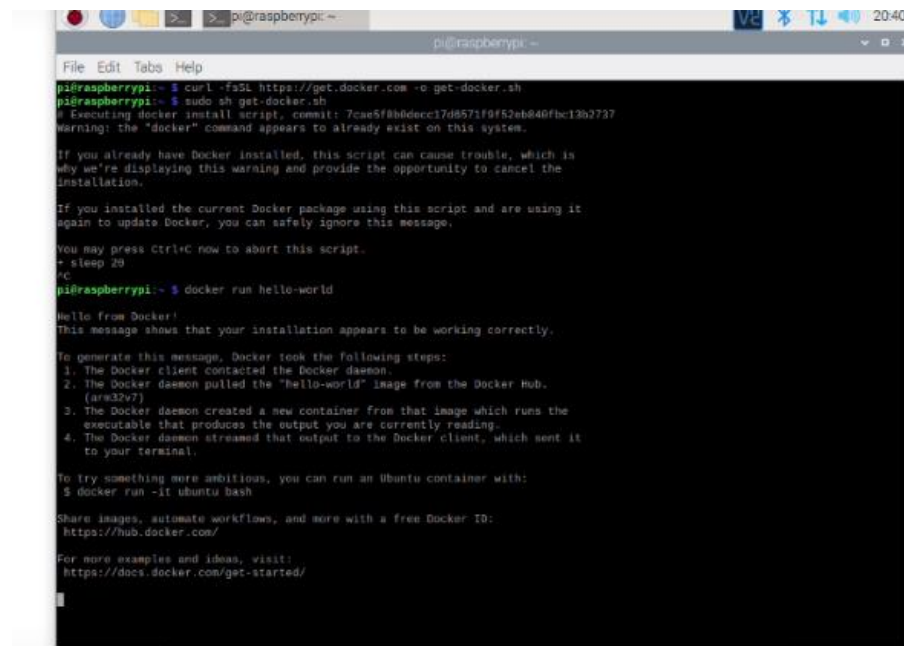


Figure 3.2.5 Running the Docker installation command on terminal

Once all resources are installed, we initialize docker through a call function to set up the interface.



```
pi@raspberrypi:~$ curl -fsSL https://get.docker.com -o get-docker.sh
pi@raspberrypi:~$ sudo sh get-docker.sh
# Executing docker install script, commit: 7cae5f8b0decc17d8571f9f52eb840fbc13b2737
Warning: the "docker" command appears to already exist on this system.

If you already have Docker installed, this script can cause trouble, which is
why we're displaying this warning and provide the opportunity to cancel the
installation.

If you installed the current Docker package using this script and are using it
again to update Docker, you can safely ignore this message.

You may press Ctrl+C now to abort this script.
^C sleep 20
^C
pi@raspberrypi:~$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (a3332v7)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

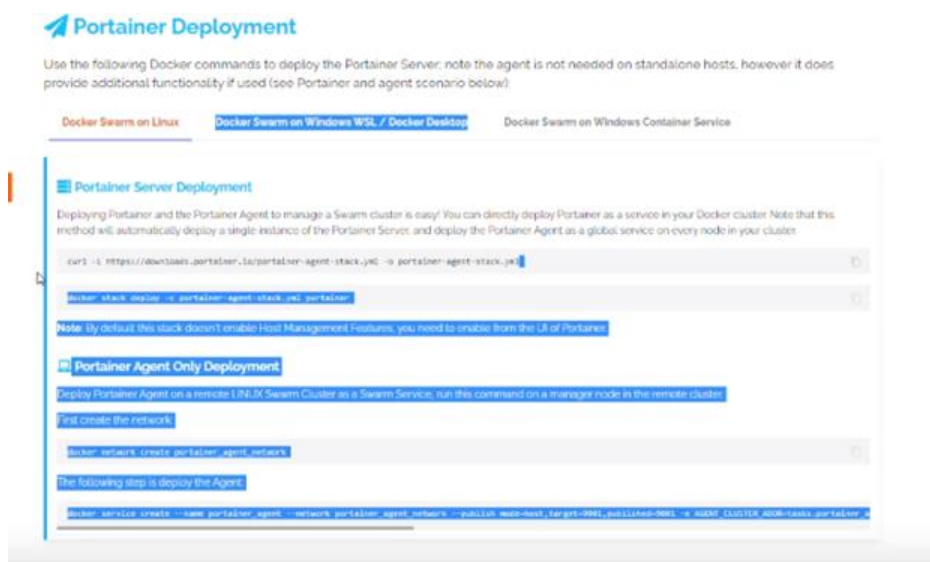
Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

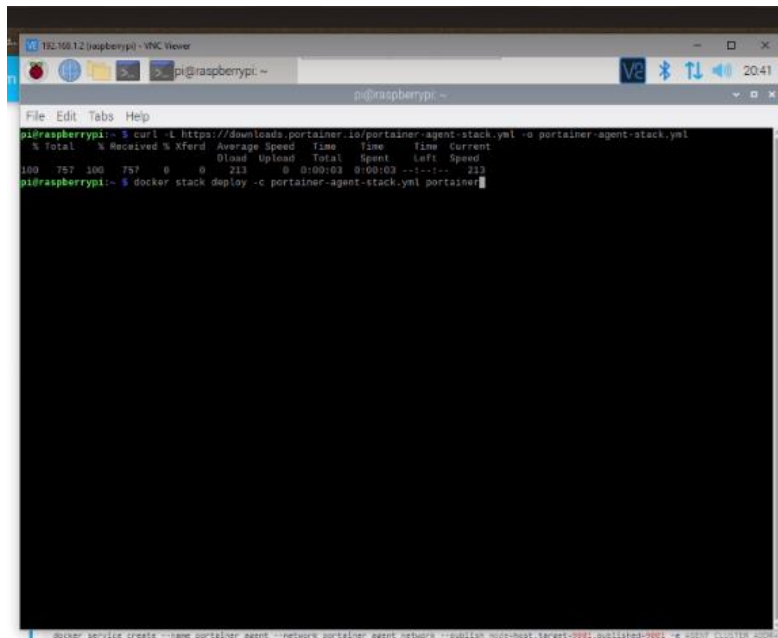
For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

When deploying a platform such as docker management is hard as it needs more resources and time to contain all the UI and programs in the container.

For that purpose we used Portainer, an UI system for making it easier for you to

manage your Docker containers, it allows you to manage containers, images, networks, and volumes from the web-based Portainer dashboard tool.





Installing the docker container through portainer and assigning a UI between the client and server platform.

This makes sure every time a new client is created a particulate container is assigned to that specific user which will be explained later.

This is where the domain of the server is set up to the server port to initiate the server along with necessary details such as IP address, Local net, ip address, memory access, dns access, and client-side interface are configured.

Step 4: Client-Side Web Page UI

Now all basic setup of the server is done to run a platform service within the website. Now we need to get a UI for the client side of the server so that each time a new login is created the data is stored within the server. We create a simple user interface on port 9000 of the system.

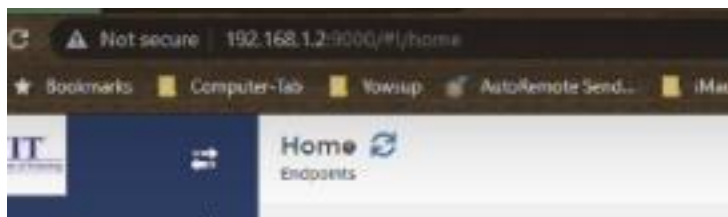


Figure 3.2.9 Here we have successfully setup a server on port 9000

This webpage is connected to docker which is programmed to create a container when a user profile is created.

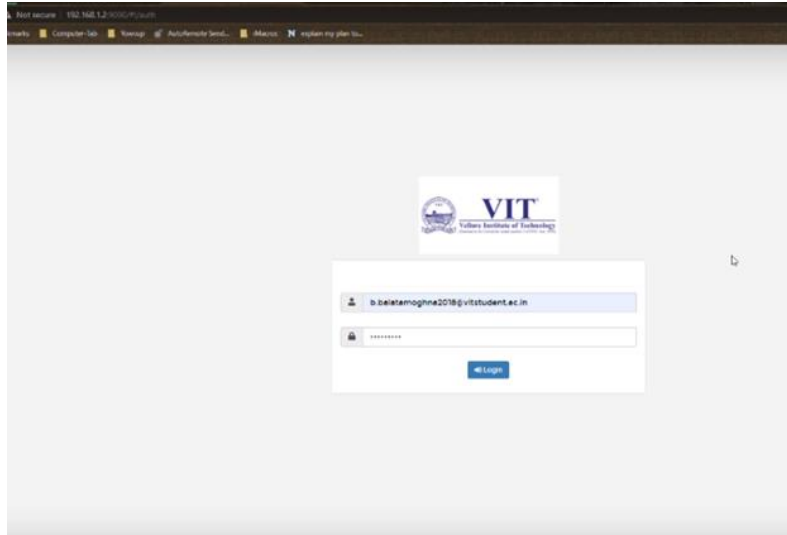


Figure 3.2.10 The Client-side Login page of the server

The port 9000 is specially chosen as this is a virtual port meaning that either does not connect to a real serial port or adds functionality to a real serial port through software extension. Developing a VM on the port is much easier and secure as it has no connection between each port of the server network, so security is high and if one problem occurs other containers are not affected.

This is running in Nodejs where the image of the user details is used for invoking the docker depository from the server side to the client side for creating the container from the copy of the image.

We establish a control over the resource within the depository and using which we assign each container a domain not so that each time it is called it always goes to the assigned port rather than some random within the portainer.

Step 5: Allocating the port for each container.

For a container to work within the VM interface it must have an Ip address and port for deploying it. When the port is assigned to that container, we create a user UI where each person can set up their system to access the resources from the main storage. But for this to work, we must assign a port for every user within the server.

For that purpose, first we need to check if the port is free or not. This is done using canyouseeme.org to detect open ports on the connection of your local Ip.

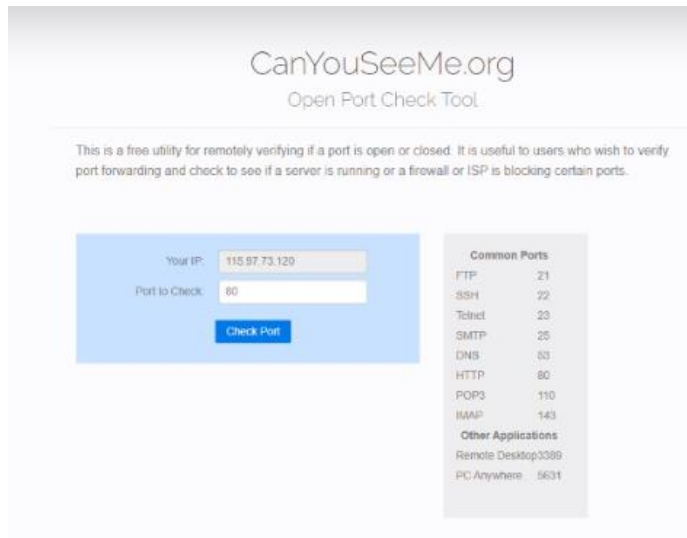


Figure 3.2.11 using canyouseeme.org to detect open ports on the connection of your local ip.

Once the port needed to host is found to be free within our network, we now need to assign the domain name for that particular port that was reserved. Each port needs a domain name and ip address for each container of the server. But this also means that when we allow a port to be allocated as public, the chances of malware occurrence increase. So, to prevent that we use Cloudflare to build a firewall on the main domain name of the server which we assigned to the website.

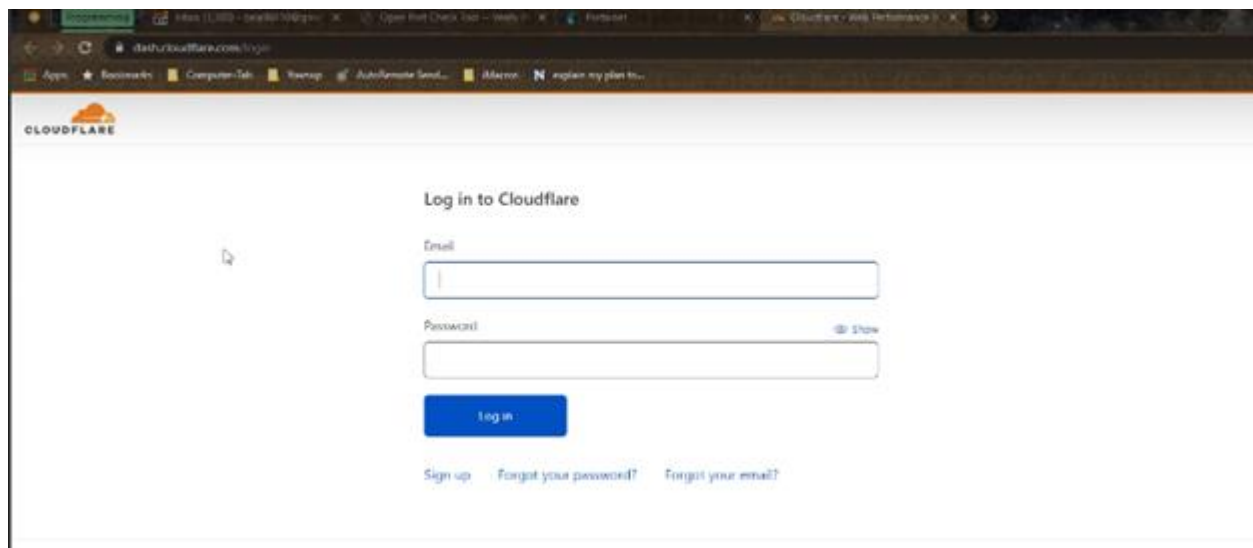


Figure 3.2.12 we use Cloudflare to build a firewall on the main domain name of the server which we assigned to the website

It protects against threats such as SQL injection and identity theft. Cloudflare also improves site performance and speeds up loading times by using their multiple data

centers that are located around the world. This allows us to use the data resource with optimal usage and less load on the server side.

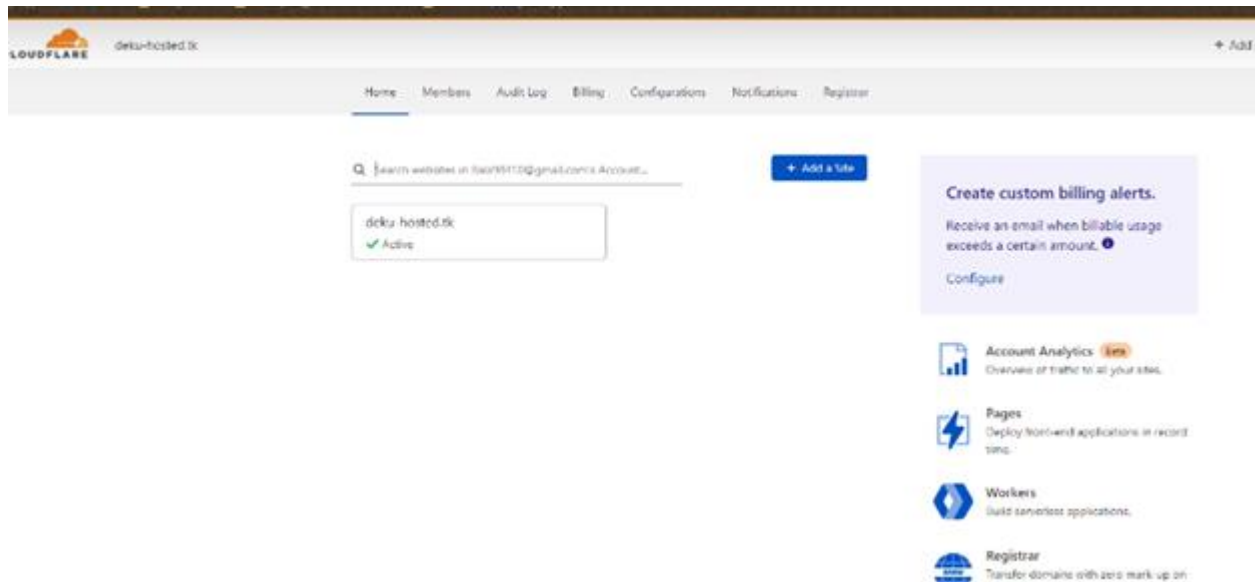


Figure 3.2.13 Cloudfare is used to improve the site performance and speeds up loading times by using their multiple data centers

We can see here the domain name associated with our server is a specific port with nginx proxy manager. Each time a port is chosen, the port is passed through this firewall for checking a virus and hides user's info from the public.

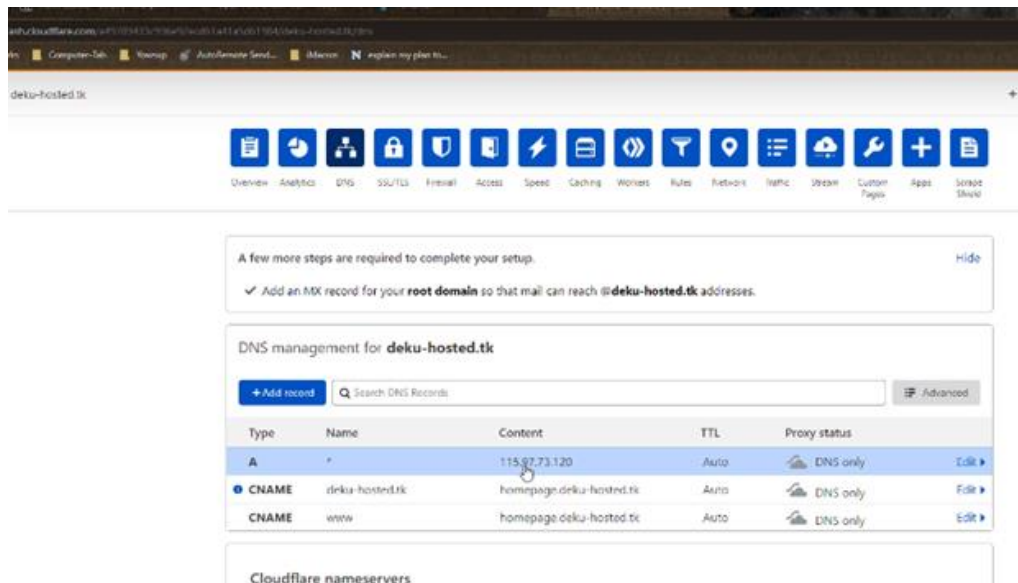


Figure 3.2.14 the port is passed through this firewall for checking a virus and hides user's info from the public.

Finally, after deploying Cloudflare the website, APIs, and applications within the server are hidden from the public and only the necessary details such as user name and data information is visible from the website.

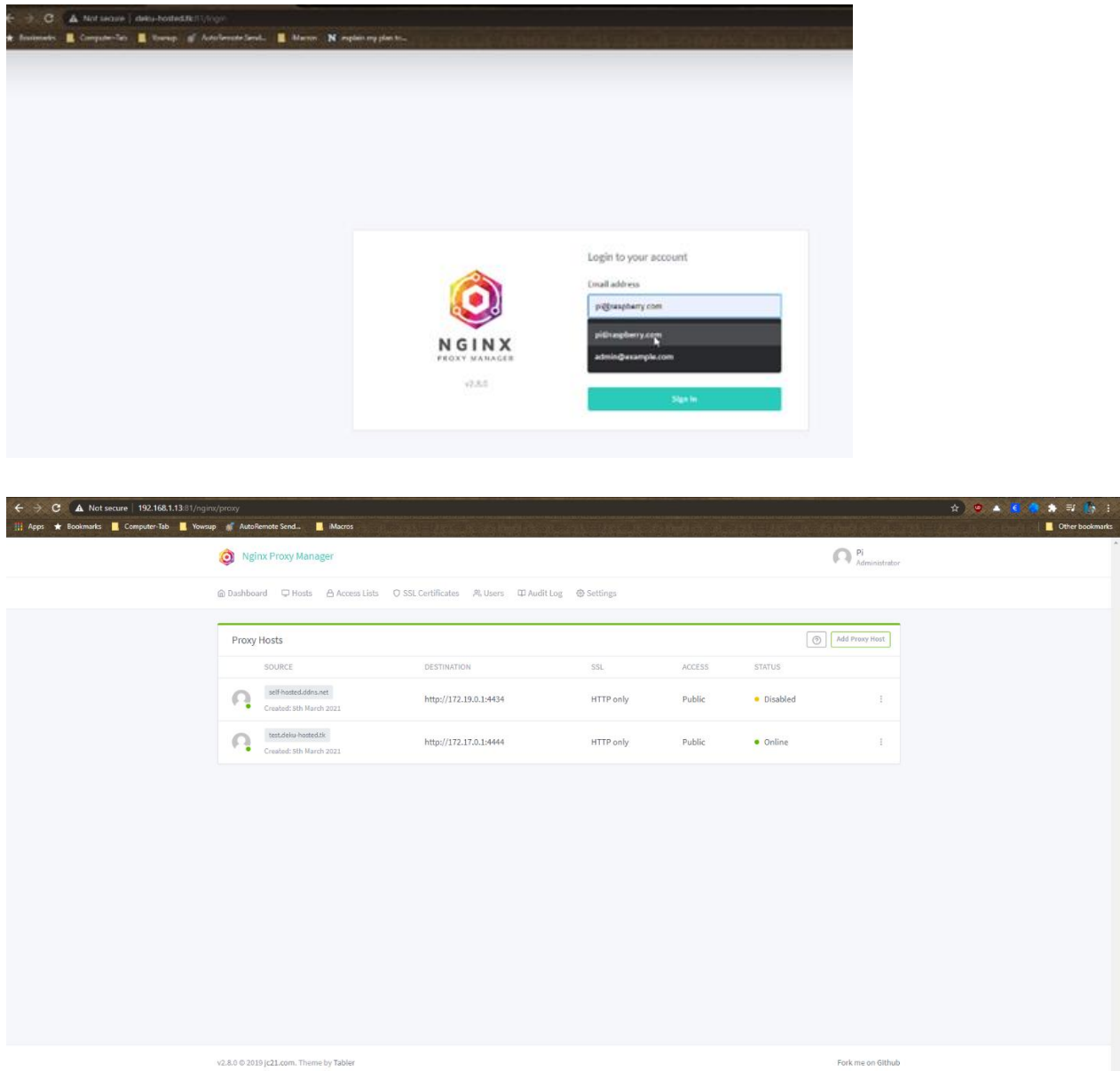


Figure 3.2.16 It shows only the necessary details such as user name and data information is visible from the website

The Nginx Proxy Manager allows for Reverse proxy managing, using this feature. We can have multiple containers directed towards different subdomains under one common domain. (Used domain here is deku-hosted.tk)

With the use of this feature. We can also give SSL certs for each domain free of cost. This ensures HTTPS connection and is secure.

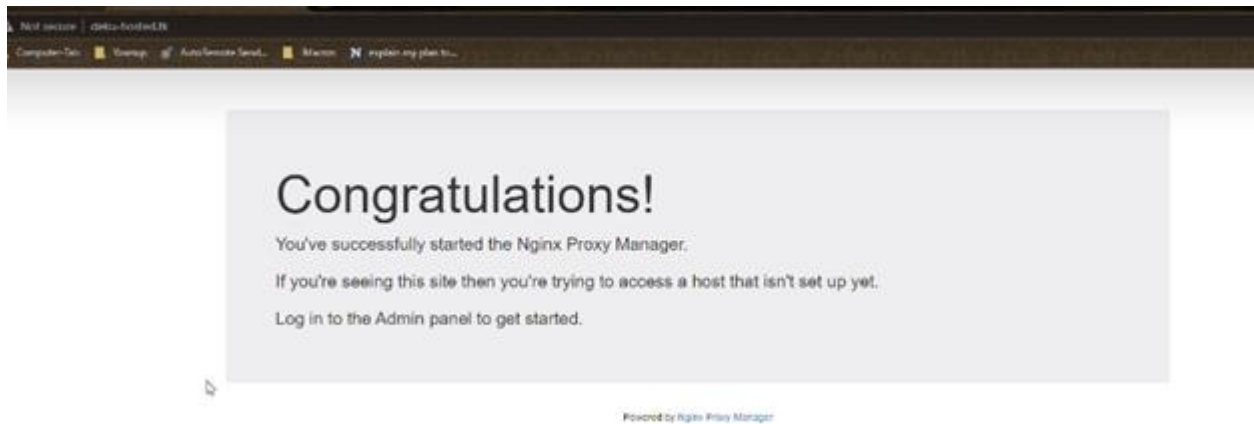


Figure 3.2.17 Here we have used Nginx Proxy Manager which allows for Reverse proxy managing

Step 6: Docker Container

Each container is configured to a set of rules per user setting which can be seen from the admin login. When a user logs into the server, a new container is created using the image of the data imputed from the user.

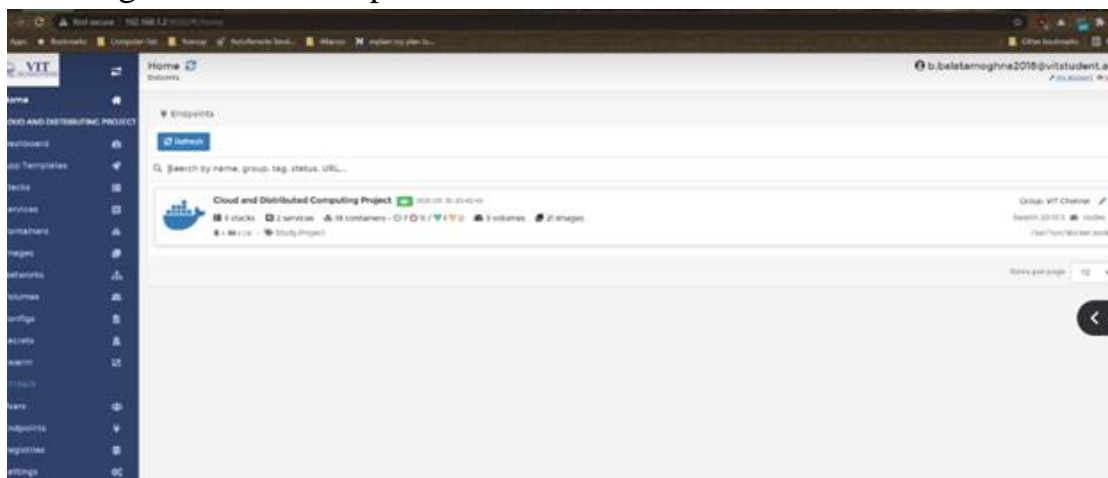


Figure 3.2.18

Docker allocates a read-write filesystem to the container, by doing so allows a running container to create or modify files and directories in its local filesystem. This also makes it visible to see the change and how that change could affect that section of the server without going through all the config file of each data image making the process much faster and easier to understand.

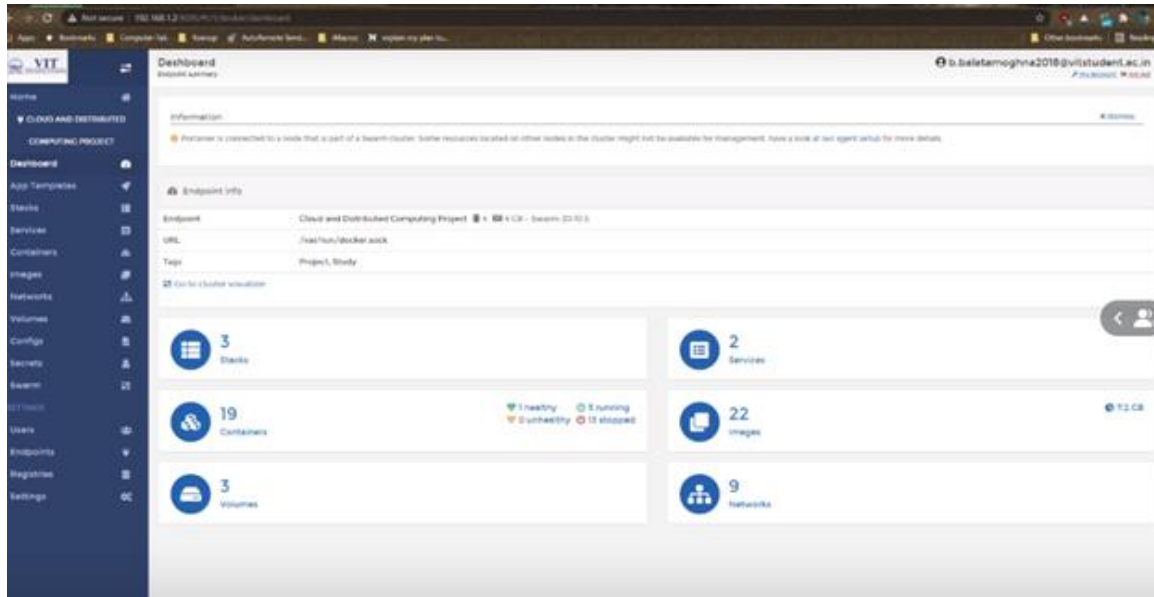


Figure 3.2.19 Here it is visible to see the change and how that change could affect that particular section of the server

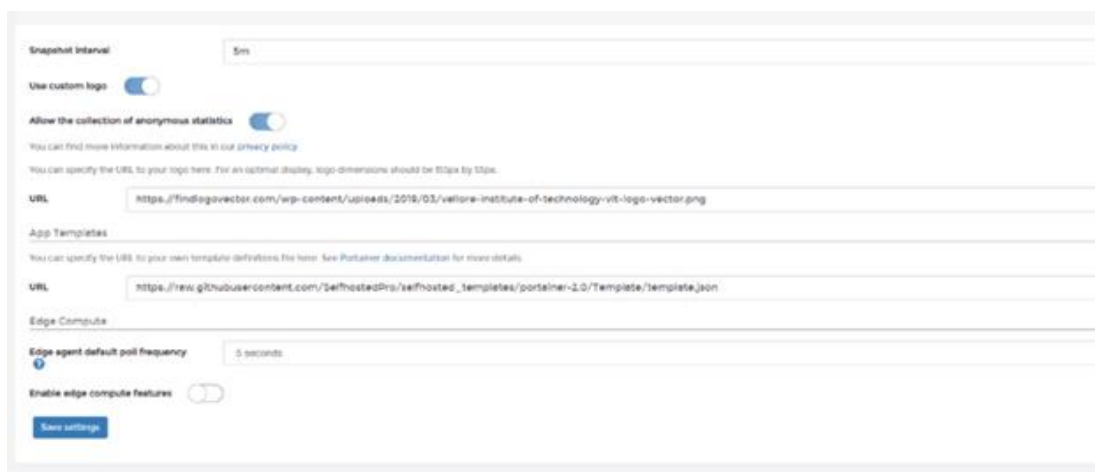


Figure 3.2.20 URL of our web server within the docker container

Each time a container is deployed, the detail of each is shown within the data cluster of the web server. This way we can manage the port for each user and allocate resources in the proper manner.

Name	State	Quick actions	Stack	Image	Created	IP Address	Published Ports	Over
nginx_app_1	healthy		nginx	nginx/nginx-proxy-manager:latest	2021-03-15 18:18:49	172.21.0.3	80:80:80	20 a
portainer_portainer.1.ukqknl...	running		portainer	portainer/portainer-ce:latest	2021-05-30 20:42:33	10.0.0.5	-	20 a
portainer_agent.osu.77da62etq...	running		portainer	portainer/agent:latest	2021-05-30 20:42:15	10.0.1.7	-	20 a
WebhookAPP	running		-	linussserve/nginx:latest	2021-05-17 10:22:58	172.17.0.4	4301:443	20 a
cloudflareddns_cloudflare-ddns_1	running		cloudflareddns	sdnu/cloudflare-ddns:latest	2021-03-16 17:08:39	172.22.0.2	-	20 a
nginx_db_1	running		nginx	yobaxsystems/nginx-mariadb:latest	2021-03-15 17:40:55	172.21.0.2	-	20 a
portainer	running		-	portainer/portainer-ce	2021-03-05 12:37:22	172.17.0.2	8000:8000	20 a
zen_lovelace	stopped		-	hello-world	2021-05-30 20:40:19	-	-	20 a
portainer_agent.osu.77da62etq...	stopped		portainer	portainer/agent:latest	2021-05-27 13:44:41	-	-	20 a
portainer_portainer.1.k3d0vc...	stopped		portainer	portainer/portainer-ce:latest	2021-05-27 13:44:36	-	-	20 a
portainer_agent.osu.77da62etq...	stopped		portainer	portainer/agent:latest	2021-05-24 22:42:24	10.0.1.3	-	20 a
portainer_portainer.1.kd2bpf...	stopped		portainer	portainer/portainer-ce:latest	2021-05-24 22:42:18	10.0.0.4	-	20 a
Backend_API	stopped		-	nodeapp-webapp	2021-05-18 11:55:16	-	-	20 a
portainer_portainer.1.xd0fex...	stopped		portainer	portainer/portainer-ce:latest	2021-05-17 09:05:09	10.0.0.5	-	20 a
portainer_agent.osu.77da62etq...	stopped		portainer	portainer/agent:latest	2021-05-17 09:05:00	10.0.1.7	-	20 a
portainer_portainer.1.c0bwgh...	stopped		portainer	portainer/portainer-ce:latest	2021-05-17 09:04:50	-	-	20 a
portainer_agent.osu.77da62etq...	stopped		portainer	portainer/agent:latest	2021-05-17 09:04:34	-	-	20 a
UI-Monitor	stopped		-	nodeapp-iot	2021-04-26 18:52:01	-	-	20 a
mysql_xing	stopped		-	nodeapp-webapp	2021-04-07 09:51:24	-	-	20 a
nginx	stopped		-	linussserve/nginx	2021-03-25 17:06:38	172.17.0.3	80:80	20 a

cloudflareddns_cloudflare-ddns_1	running		cloudflareddns	sdnu/cloudflare-ddns:latest	2021-03-16 17:08:39	172.22.0.2	-	20
nginx_db_1	running		nginx	yobaxsystems/nginx-mariadb:latest	2021-03-15 17:40:55	172.21.0.2	-	20
portainer	running		-	portainer/portainer-ce	2021-03-05 12:37:22	172.17.0.2	8000:8000	20
zen_lovelace	stopped		-	hello-world	2021-05-30 20:40:19	-	-	20

Figure 3.2.21, the detail of each container is shown within the data cluster of the web server

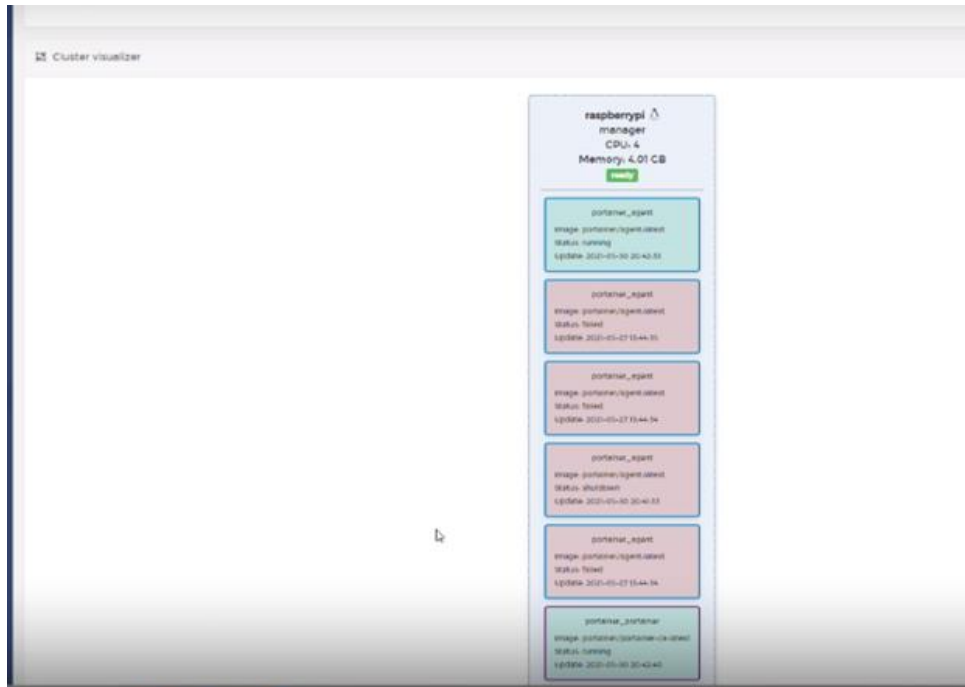


Figure 3.2.22 the detail of each is shown within the data cluster of the web server

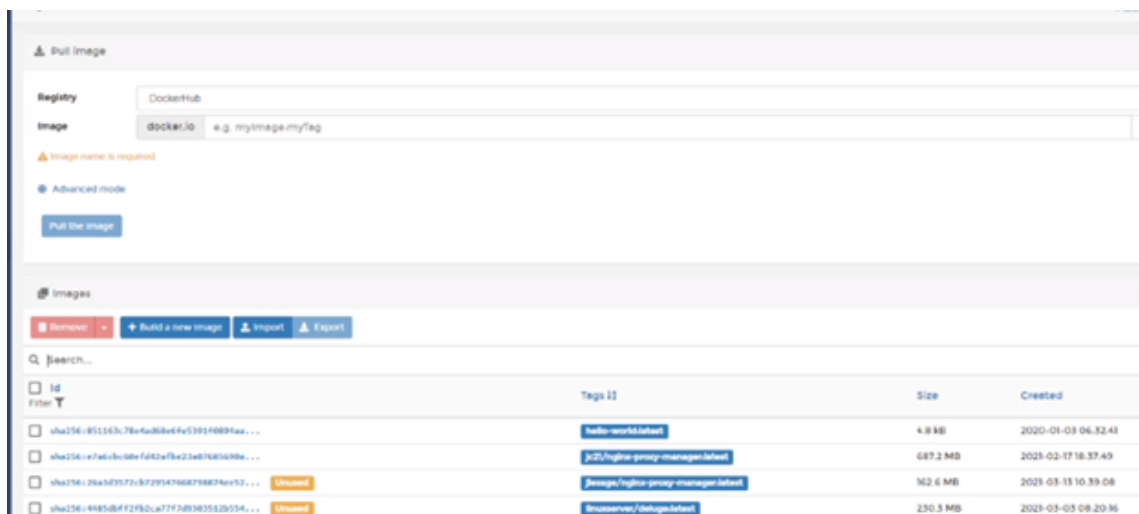


Figure 3.2.23 Shows us the container created using the image imported into docker using the URL

For each container a specific port's address is assigned, within that ip address of that port the data of individual users is stored where any data can be modified as per user requirement within their personnel port ip of the container.

Each user detail can be seen from the Rpi terminal with a specific port ip address assigned. Each task such as login, change in data, deletion/creation can be viewed as well.

```

File Edit Tabs Help
pi@raspberrypi:~$ curl -L https://downloads.portainer.io/portainer-agent-stack.yml -o portainer-agent-stack.yml
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 757 100 757 0 0 213 0 0:00:03 0:00:03 --:--:-- 213
pi@raspberrypi:~$ docker stack deploy -c portainer-agent-stack.yml portainer
Updating service portainer_agent (id: 9o9wn3l69a7bbevlgo0qgbqtw)
Updating service portainer_portainer (id: 5txtvh0dulmuv3wbxxelyk3o7)
pi@raspberrypi:~$ docker ps
CONTAINER ID IMAGE NAMES COMMAND CREATED STATUS PORTS
38219c650278 portainer/portainer-ce:latest "/portainer -H tcp://..." 4 minutes ago Up 4 minutes 8000/tcp
00/tcp portainer_portainer.1.1u4q4nlzfjd1apbm6qe5njx0d
3d620d19b04f portainer/agent:latest "/agent" 5 minutes ago Up 4 minutes
portainer_agent.ovu97da62ezsj9yhkjr0nj6lu.782toi0se0w00mgngb008xqu6
b58591416fb9 linuxserver/nginx:latest "/init" 13 days ago Up 3 days 0.0.0.0:4017
2->80/tcp, 0.0.0.0:40171->443/tcp Webhost7ARP
09c46604712e oznu/cloudflare-ddns:latest "/init" 2 months ago Up 3 days
cloudflareddns_cloudflare-ddns_1
6e8ff7e5e1d8 jc21/nginx-proxy-manager:latest "/init" 2 months ago Up 3 days (healthy) 0.0.0.0:80-8
1->80-81/tcp, 0.0.0.0:443->443/tcp npman_app_1
383c0fc9c19f yobasystems/alpine-mariadb:latest "/scripts/run.sh" 2 months ago Up 3 days 3386/tcp
npman_db_1
15e1d2e259d7 portainer/portainer-ce "/portainer" 2 months ago Up 3 days 0.0.0.0:8000
->8000/tcp, 0.0.0.0:9000->9000/tcp portainer
pi@raspberrypi:~$

```

Figure 3.2.24 The user detail can be seen from the Rpi terminal with a specific port ip address assigned

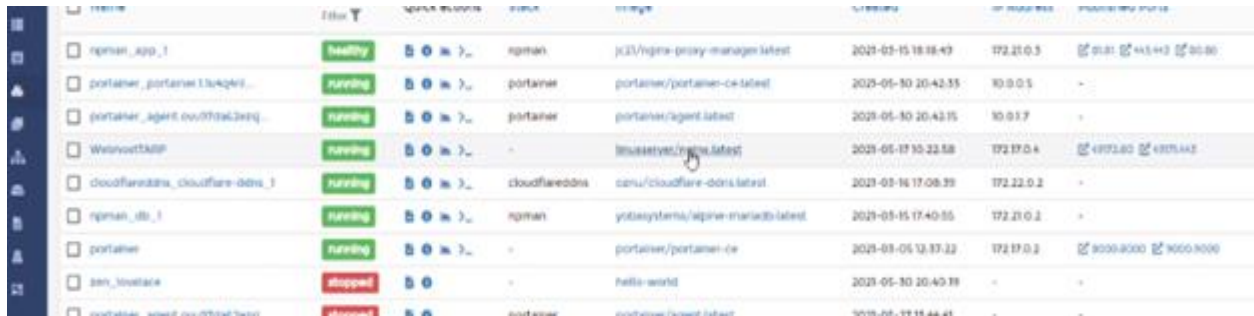
Step 7: Server Performance, Security, and data allocation

We created the container from the imported image from the user and a VM is created to run at a specific port. But as the number of users increases the server resources get limited, making it less efficient and more power consuming causing lags in the server.

To counteract this, we used nginx proxy manager which we assigned. It is built to offer low memory usage and high concurrency with no minimal loss on the performance of the server and excels at serving static content quickly and is designed to pass dynamic requests off to other software. This way we can organize each port container at high efficiency and control the data deposited. This also helps us to run multiple instances with a single port in multiple virtual containers with minimal loss in data usage and resource sharing in a more effective way. Also, in terms of real-world use-cases, Nginx can handle in which each server handles requests for static and dynamic content.

Once all configurations are done, any request from the client is passed from the nginx through the reverse proxy of the server client.

To get the reverse proxy to work, we need to reload the nginx service inside the container. From the host, run `docker exec <container-name> nginx -t`. This will run a syntax checker against your configuration files. This should output that the syntax is ok. Now run `docker exec <container-name> nginx -s reload`. This will send a signal to the nginx process that it should reload. When this is done now, we have a running reverse proxy, and should be able to access your server from it.



Name	Status	Ports	Image	Created	Updated	Restart Policy
nginx_app_1	healthy	5.0.0.1	nginx	2023-05-15 18:18:43	172.21.0.5	no
portainer_portainer_1	running	5.0.0.1	portainer	2023-05-10 20:42:55	10.0.0.1	no
portainer_agent_007f0a32eq	running	5.0.0.1	portainer	2023-05-10 20:42:15	10.0.1.7	no
WinevotKMP	running	5.0.0.1	linuxserver/nginx	2023-05-17 10:22:58	172.17.0.4	no
cloudflaredns_cloudflare-dns_1	running	5.0.0.1	cloudflaredns	2023-05-16 17:08:39	172.22.0.2	no
nginx_db_1	running	5.0.0.1	nginx	2023-05-15 17:40:55	172.21.0.2	no
portainer	running	5.0.0.1	portainer/portainer-ce	2023-05-05 12:37:22	172.17.0.2	no
api_hostapi	stopped	5.0.0.1	hello-world	2023-05-10 20:40:18	-	no

Figure 3.2.25

As you can see, the server.config file consists of two parts. An upstream part and a server part. In server part this is where you are defining the port receiving the incoming requests, what domain this configuration should match, and where it should be sent to. The way this server is being setup; you should make a file for each service that you want to proxy requests to so that you need some way to distinguish which file to receive each request. We used the port address as the destination for distinguishing which file is to be sent to what port and which resource as it be allocated with it.

Step 8: Testing the server with load.

We used a data image from one of the projects to check the performance of the client.

As mentioned above we followed all steps from data importing to deploying a container on a separate port on the client side.

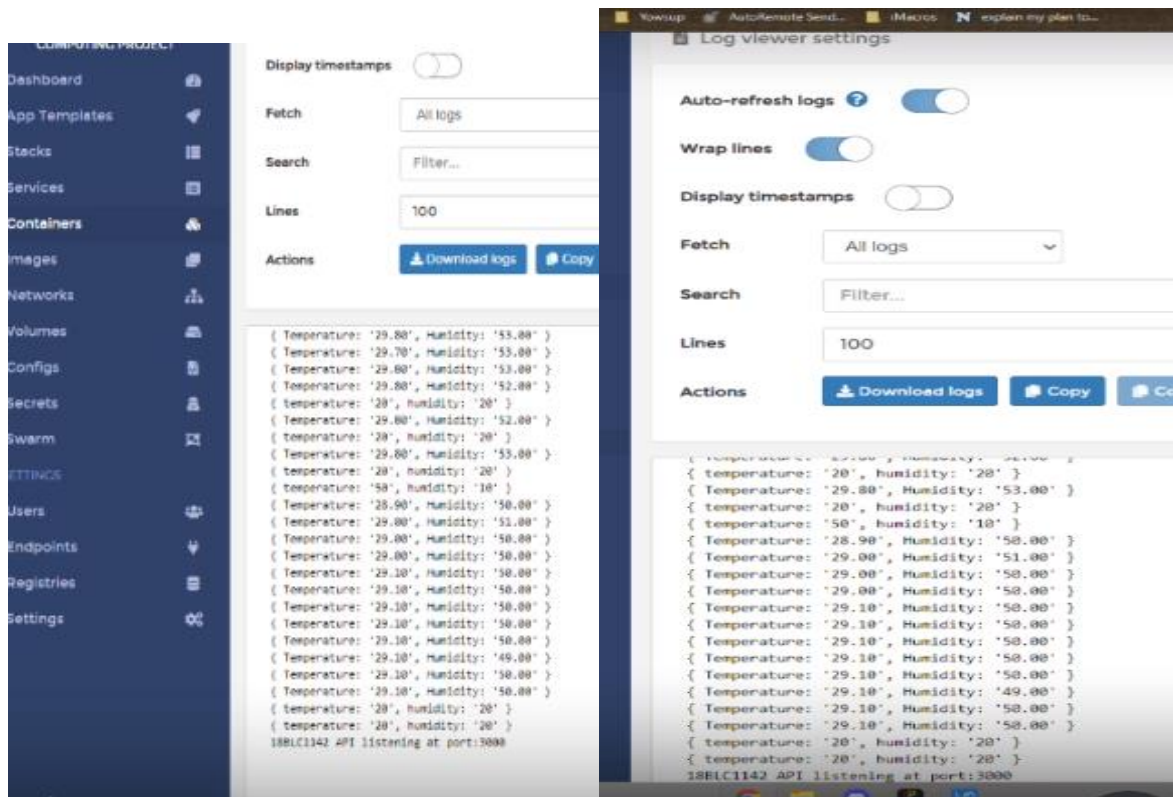



Figure 3.2.26 Here we observe the server load and data output through the portainer

We were able to observe the output of the project within the server client using a newly created user domain at port 3000.

Users  User management b.balatamoghne2

[+ Add a new user](#)

Username

Password

Confirm password

Administrator ☐

Add to team(s) [Select one or more teams](#)

[+ Create user](#)

Note: non-administrator users who aren't in a team don't have access to any endpoints by default. Head over to the [Endpoints view](#) to manage their accesses.

Users

[Refresh](#)

Search...

<input type="checkbox"/> Name	Role	Authentication
<input type="checkbox"/> aditya.jaganath2018@vitstudent.ac.in	team leader	Internal
<input type="checkbox"/> anish.subramanian2018@vitstudent.ac.in	user	Internal
<input type="checkbox"/> b.balatamoghna2018@vitstudent.ac.in	administrator	Internal
<input type="checkbox"/> s.vandeeswaran2018@vitstudent.ac.in	user	Internal

Figure 3.2.27 Show which members of the port are associated with that ip address alongside the data they saved within the server.

Search...					
<input type="checkbox"/> Name	Image	Scheduling Mode	Published Ports	Last Update	Ownership
<input type="checkbox"/> portainer_agent	portainer/agent:latest	global 1 / 1	-	2021-05-30 20:42:40	administrators
<input type="checkbox"/> portainer_portainer	portainer/portainer-ce:latest	replicated 1 / 1 Scale	8000-8000 9000-9000	2021-05-30 20:42:45	administrators
Items per page 10					

CODING AND ANALYSIS

```
#include <ESP8266HTTPClient.h>

#include <ESP8266WiFi.h>;

#include <WiFiClient.h>;

#include <DHT.h>;

#define DHTPIN 0

DHT dht(DHTPIN, DHT11);

const unsigned char buzz =14;

const char* ssid = "Madaya";

const char* password = "1112223333";

int val;

int LDRpin = A0; //LDR Pin Connected at A0 Pin

String serverName = "http://iothealth.ddns.net:1142/values/";

WiFiClient client;

String apiKey = "197GD6ZDZT56AZM6";

unsigned long myChannelNumber = 1123344;

unsigned long lastTime = 0;

// Timer set to 10 minutes (600000)

//unsigned long timerDelay = 600000;

// Set timer to 5 seconds (5000)

unsigned long timerDelay = 5000;


void setup()
```

```

{
  Serial.begin(9600);
  delay(10);
  dht.begin();
  WiFi.begin(ssid, password);
  pinMode(buzz,OUTPUT);
}

void loop()
{
  val = analogRead(LDRpin);
  Serial.print(val);
  delay(1000);
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  //Send an HTTP POST request every 10 minutes
  if ((millis() - lastTime) > timerDelay) {
    //Check WiFi connection status
    if(WiFi.status()== WL_CONNECTED){
      HTTPClient http;

      String serverPath = serverName + "Temperature="+t+"&Humidity="+h;

```

```
// Your Domain name with URL path or IP address with path
http.begin(serverPath.c_str());

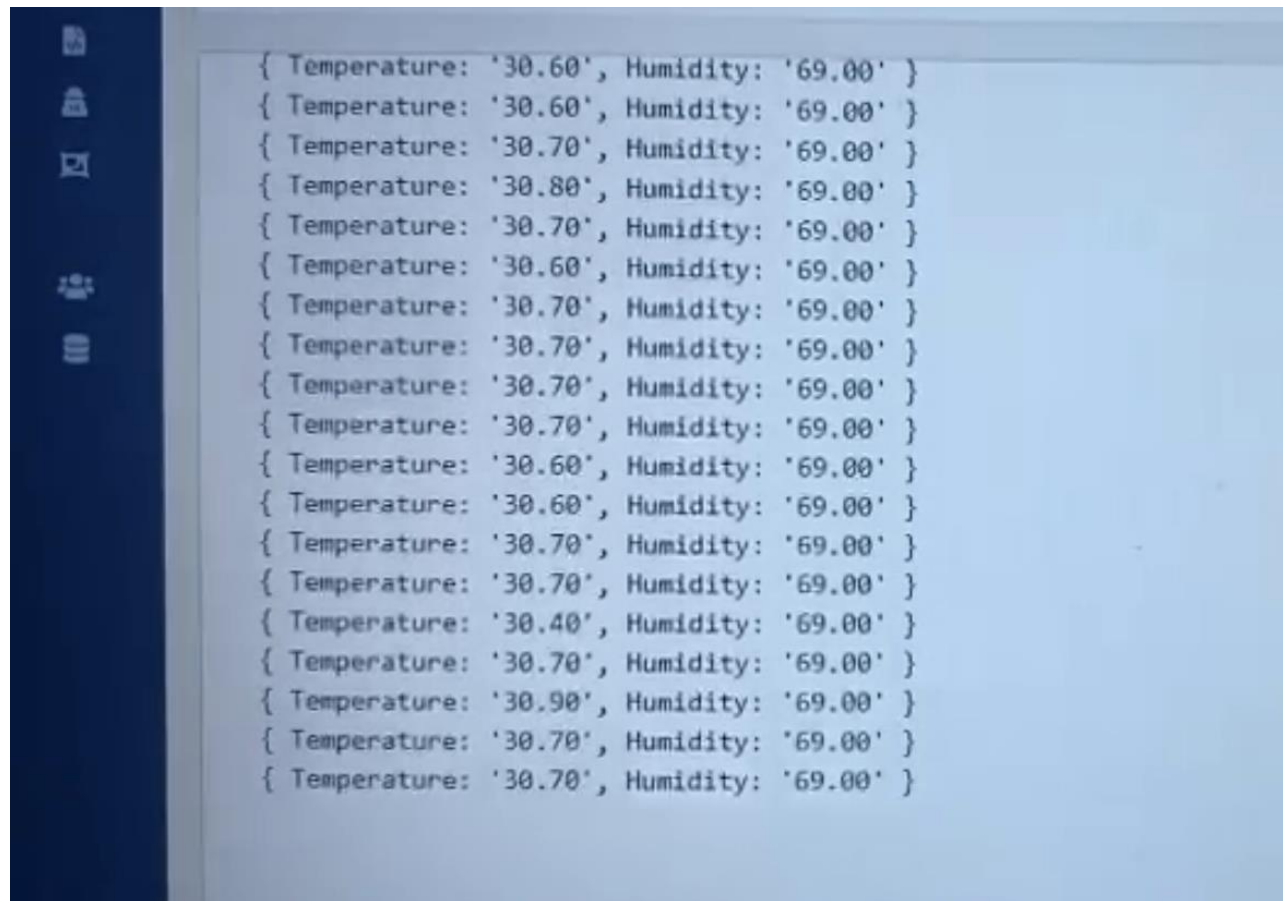
// Send HTTP GET request
int httpResponseCode = http.GET();

if (httpResponseCode>0) {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
    String payload = http.getString();
    Serial.println(payload);
}
else {
    Serial.print("Error code: ");
    Serial.println(httpResponseCode);
}

// Free resources
http.end();
}else {
    Serial.println("WiFi Disconnected");
}

lastTime = millis();
}
}
```

4.3 RESULT AND ANALYSIS:



The image shows a screenshot of a web application interface. On the left, there is a dark blue sidebar with several white icons: a document, a bell, a square, a person, and a list. The main content area is light blue and displays a list of 20 data points. Each data point is a JSON object with 'Temperature' and 'Humidity' keys. The values for Temperature range from '30.40' to '30.90', and the values for Humidity are all '69.00'.

{ Temperature: '30.60', Humidity: '69.00' }
{ Temperature: '30.60', Humidity: '69.00' }
{ Temperature: '30.70', Humidity: '69.00' }
{ Temperature: '30.80', Humidity: '69.00' }
{ Temperature: '30.70', Humidity: '69.00' }
{ Temperature: '30.60', Humidity: '69.00' }
{ Temperature: '30.70', Humidity: '69.00' }
{ Temperature: '30.70', Humidity: '69.00' }
{ Temperature: '30.70', Humidity: '69.00' }
{ Temperature: '30.70', Humidity: '69.00' }
{ Temperature: '30.60', Humidity: '69.00' }
{ Temperature: '30.60', Humidity: '69.00' }
{ Temperature: '30.70', Humidity: '69.00' }
{ Temperature: '30.70', Humidity: '69.00' }
{ Temperature: '30.40', Humidity: '69.00' }
{ Temperature: '30.70', Humidity: '69.00' }
{ Temperature: '30.90', Humidity: '69.00' }
{ Temperature: '30.70', Humidity: '69.00' }
{ Temperature: '30.70', Humidity: '69.00' }

5.1 CONCLUSION AND FUTURE WORK

Hence, we have successfully built a self-hosted cloud server independently running on a raspberry pi with the help of docker through Portainer.

Future work

We can scale the project up dynamically to handle the traffic better. We can also improve the security feature of the cloud services and finally we can open testing to the public and can note the performance of the cloud service and find ways to improve the performance.

5.2 COST ANALYSIS

SOFTWARE	Usage	Cost
JavaScript	Creating server and user client sides (Web page), storage of data, controls and manages all request and access for user	Free open-source
Docker	It runs your application inside a container, uses minimum resources, can be deployed faster, and it can scale quickly.	Free open-source
Portainer	It acts a medium between docker and the user it makes it easier to access docker without any prior knowledge	Free open-source
NGINX	used as a reverse proxy and load balancer to manage incoming traffic and distribute it to slower upstream servers	Free open-source
Node.js	It is primarily used for non-blocking, event-driven servers, due to its single-threaded nature. It's used for traditional web sites and back-end API services	Free open-source
Raspberry Pi 4, 4GB RAM	The Raspberry Pi is a low cost, credit-card sized computer we have used it to host the server	₹ 6,500.00

REFERENCE

- S. Wang, L. Zhu and M. Cheng, "Docker-based Web Server Instructional System," 2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS), 2019, pp. 285-289, doi: 10.1109/ICIS46139.2019.8940219.
- D. Huang, H. Cui, S. Wen and C. Huang, "Security Analysis and Threats Detection Techniques on Docker Container," *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*, 2019, pp. 1214-1220, doi: 10.1109/ICCC47050.2019.9064441.
- H. Zeng, B. Wang, W. Deng and W. Zhang, "Measurement and Evaluation for Docker Container Networking," 2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2017, pp. 105-108, doi: 10.1109/CyberC.2017.78.
- D. Huang, H. Cui, S. Wen and C. Huang, "Security Analysis and Threats Detection Techniques on Docker Container," 2019 IEEE 5th International Conference on Computer and Communications (ICCC), 2019, pp. 1214-1220, doi: 10.1109/ICCC47050.2019.9064441.
- A. Gopalasingham, D. G. Herculea, C. S. Chen and L. Roullet, "Virtualization of radio access network by Virtual Machine and Docker: Practice and performance analysis," 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 2017, pp. 680-685, doi: 10.23919/INM.2017.7987358.
- A. Krasnov, R. R. Maiti and D. M. Wilborne, "Data Storage Security in Docker," 2020 SoutheastCon, 2020, pp. 1-1, doi: 10.1109/SoutheastCon44009.2020.9249757.
- A. Jaison, N. Kavitha and P. S. Janardhanan, "Docker for optimization of Cassandra NoSQL deployments on node limited clusters," 2016 International Conference on Emerging Technological Trends (ICETT), 2016, pp. 1-6, doi: 10.1109/ICETT.2016.7873643.
- Preeth E N, F. J. P. Mulerickal, B. Paul and Y. Sastri, "Evaluation of Docker containers based on hardware utilization," 2015 International Conference on Control Communication & Computing India (ICCC), 2015, pp. 697-700, doi: 10.1109/ICCC.2015.7432984.
- W. Yang, X. Tan, J. Guo and S. Wang, "The vulnerability analysis and security enhancement of Docker", *Information Security and Technology*, vol. 7, no. 4, pp. 21-23, 2016.
- J. Liu, "Analysis of Docker virtualization technology", *Security & Informatization*, pp. 73-81, Jan. 2019.

- Molnar, David & Schechter, Stuart. (2010). Self Hosting vs. Cloud Hosting: Accounting for the security impact of hosting in the cloud.
- <https://www.c-sharpcorner.com/article/how-to-build-personal-web-api-server-using-raspberry-pi-and-nodejs/> - “How To Build Personal Web API Server Using Raspberry PI And Node.js”
- <https://dev.to/teamallnighter/i-built-a-simple-rest-api-with-node-and-a-raspberry-pi-m7a> - “I built a simple rest API with node and a Raspberry Pi”
- https://www.reddit.com/r/selfhosted/comments/gwhrqv/how_to_protect_your_self_hosted_architecture/ - “How to protect your self hosted architecture against DDOS ?”
- <https://www.nginx.com/blog/mitigating-ddos-attacks-with-nginx-and-nginx-plus/> - “Mitigating DDoS Attacks with NGINX and NGINX Plus”