ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"DESARROLLO DE UN PROTOTIPO DE SISTEMA DE TRANSMISIÓN DE BAJA POTENCIA PARA MEDICIÓN DE HUMEDAD Y TEMPERATURA EN UN CENTRO DE DATOS, USANDO EL PROTOCOLO LORA"

EXAMEN DE GRADO

PREVIO A LA OBTENCIÓN DEL TÍTULO DE:

MAGISTER EN TELECOMUNICACIONES

AUTOR:
JUAN ANDRÉS MAROTO LEMA

GUAYAQUIL – ECUADOR
AÑO 2020

AGRADECIMIENTO

A Dios, por darme salud, por sus bendiciones y ayudarme a mantener mi fe intacta.

A mi esposa Ing. Johanna Pacheco por su apoyo incondicional, por su comprensión durante este tiempo de estudios, por ayudarme a perseverar, por darme aliento para continuar mejorando día a día, por empujarme a terminar esta meta que la había postergado y sobretodo por creer en mí.

A mis suegros porque estuvieron con mi familia, cuando yo estuve en clases o preparándome para algún examen.

DEDICATORIA

A mis maravillosos hijos Johan Andrés y Thiago Benjamín, porque hacen de mí una mejor persona.

A mi esposa Johanna, por su amor y porque sin ella no lo hubiera logrado.

A mi amada madre Arq. Gloria Lema por sus consejos y valores que inculcó en mí.

A mis hermanos Miguel y Joselyne, para que nunca dejen de soñar y creer en ellos mismos.

TRIBUNAL DE EVALUACIÓN

Ph.D. MARÍA ANTONIETA ÁLVAREZ
PROFESOR EVALUADOR

M.Sc. ALFREDO NÚÑEZ UNDA.
PROFESOR EVALUADOR

DECLARACIÓN EXPRESA

“La responsabilidad y la autoría del contenido de este Trabajo de Titulación, me corresponde exclusivamente; y doy mi consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual”.

MAROTO LEMA JUAN ANDRÉS

RESUMEN

El presente proyecto consiste en medir la temperatura y humedad en un rack, desarrollando un pequeño prototipo que transmite datos usando un sistema de baja potencia y el protocolo LORA.

El rack en el que se trabajará está ubicado en un cuarto de racks que pertenece a un centro de datos de una empresa de telecomunicaciones de Ecuador, los equipos usados para las mediciones se colocaron dentro del cuarto de rack. El LORA SHIELD con el sensor fueron puestos en la puerta del rack y el Gateway en el mismo cuarto, pero separado con una distancia de 12 metros conectados de forma inalámbrica.

El prototipo del proyecto se armó por partes y al final se implementó en el rack autorizado por la jefatura del centro de datos.

Una vez instalado el prototipo, se comenzó a recolectar los datos sensados y se comparó con los datos que estaban configurados en el equipo de enfriamiento.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ABREVIATURAS Y SIMBOLOGÍA

CSS: Chirp Spread Spectrum/ Chirp de espectro ensanchado

LPWAN: Low Power Wide Area Network/ Red de área amplia de bajo consumo

mA: Miliamperios

HR: Humedad relativa

Thingspeak: API de código abierto para el internet de las cosas

API: Interfaz de programación de aplicaciones

Kernel: Núcleo fundamental del sistema operativo

NUC: Next Unit of Computing

UMA: Unidad Manejadora de Aire

INTRODUCCIÓN

En la actualidad las empresas en el Ecuador usan equipos que generan calor, las pequeñas empresas en ocasiones no cuentan con equipos para ventilar esos lugares y por otro lado tenemos las empresas más grandes que si tienen sistemas de enfriamiento, pero muchas veces no funcionan como deberían y eso provoca que haya algún tipo de daño en los equipos sea por sobrecalentamiento o algún conato de incendio.

Por expuesto anteriormente recomendaría este prototipo, adicionando que el costo no es muy elevado si analizamos con otros dispositivos y con el tiempo de vida útil, estos son factores importantes que analizan las empresas.

El desarrollo de este trabajo consiste en cuatro partes, en la primera parte conoceremos más a fondo la problemática detectada, en el segundo capítulo desarrollaremos el prototipo de medición, en el tercer capítulo se analizará los resultados obtenidos y finalmente se concluirá si el producto es viable o no.

CAPÍTULO 1

GENERALIDADES

1.1. DESCRIPCIÓN DEL PROBLEMA

En este centro de datos existen diferentes sistemas de monitoreo de equipos, pero no tiene un monitoreo que mida la calidad de aire que llega a los racks, sea la misma que sale del equipo de enfriamiento.

Se ha pensado en esta problemática, debido que existió una detección de forma oportuna por el personal que estaba dando rondas en el sitio y se dieron cuenta del recalentamiento de un equipo.

1.2. SOLUCIÓN PROPUESTA

Con la finalidad de proteger los equipos de la humedad y conocer si la temperatura que les llega es la adecuada para su correcto funcionamiento, se creará un prototipo de sistema de medición de temperatura y humedad usando un microcontrolador LoRa Shield[1].

Para solucionar el problema, se colocará un sensor en el microcontrolador LoRa Shield, el mismo que se encargará de tomar las métricas de humedad y temperatura del rack, estas mediciones se las comparará con la información configurada de los equipos de enfriamiento ubicados en la parte externa del cuarto de rack.

La solución que se ha considerado es: conectar la Lora Shield con el sensor adaptado, a un Gateway en forma inalámbrica, esto será ubicado en un rack de tal manera que hará las lecturas de las mediciones y luego esta información será enviada a una base de datos portable, que será almacenada en un Raspberry pi, usando una transmisión de baja potencia.

El alcance de la solución será poder mostrar gráficamente en un visor web open source, las métricas de humedad y temperatura con respecto al tiempo, que le llega al rack, de tal manera que los equipos siempre estén operativos y no sean afectadas las métricas antes indicadas, se espera que no habrá interferencia, ya que los equipos que se encuentran operando en el rack están conectados de

manera alámbrica, mientras que este prototipo funcionará de manera inalámbrica.

1.3. OBJETIVOS

1.3.1. OBJETIVO GENERAL

Desarrollar un prototipo de sistema de transmisión de baja frecuencia que mida la humedad y temperatura del rack de un centro de datos, usando un sensor y el protocolo lora.

1.3.2. OBJETIVOS ESPECÍFICOS

- Realizar la lectura de las métricas de temperatura y humedad.
- Crear una base con la data de las mediciones de temperatura y humedad.
- Enviar la data desde el chip lora hasta la base de datos.
- Mostrar los resultados de los datos obtenidos en un visor web de software libre.

1.4. MARCO TEÓRICO

1.4.1. INTERNET DE LAS COSAS

Actualmente, muchos dispositivos y cosas de uso diario están trabajando bajo el principio de internet de las cosas, es un concepto que ha revolucionado las industrias con el uso de sensores y que además es muy amplio. Suelen decir que se puede conectar centenares de equipos y procesar miles de datos, dependiendo solo del router o proveedor de internet contratado, sin participación de ninguna persona que vigile en tiempo real el funcionamiento de un dispositivo o la recepción de datos en forma autónoma y no manual como se la hacía anteriormente. [2]

1.4.2. LORA Y LORAWAN

Lora es una tecnología inalámbrica, como WiFi, Bluetooth, entre otras. Funciona por modulación de radiofrecuencia, denominada CSS su propietario es Semtech.[3]

Principales ventajas de Lora:

- Tolera las interferencias
- Es muy sensible a la recepción de datos (-168dB)
- La batería puede durar hasta 10 años
- Trabaja a distancias de 10km a 20km
- Transfiere datos de hasta 255 bytes
- Usa el tipo de conexión “punto a punto”
- Opera a 915 Mhz en América

Esta tecnología es usada en conexiones de grandes distancias, en dispositivos que trabajen con internet de las cosas y que usen diferentes sensores. [3]

LoraWAN es un protocolo de red para implementación en LPWAN, sirve para comunicar y administrar dispositivos Lora. Está compuesto por gateways y nodos. [3]

- Gateways: Antenas que sirven para enviar y recibir información usando nodos.
- Nodos: Dispositivos finales que se comunican bidireccionalmente con el gateway.

El estándar LoraWAN nos permite conectar sensores, a continuación, se detallan algunas ventajas:

- Usa encriptación lo que hace que las conexiones bidireccionales sean seguras
- La batería puede durar hasta 10 años
- Opera a largas distancias hasta 20 km
- Permite conectar muchos sensores y equipos en cualquier red
- Trabaja a baja frecuencia para transmisión y servicios de localización

- Se puede interconectar con diversas redes LoraWAN por su interoperabilidad

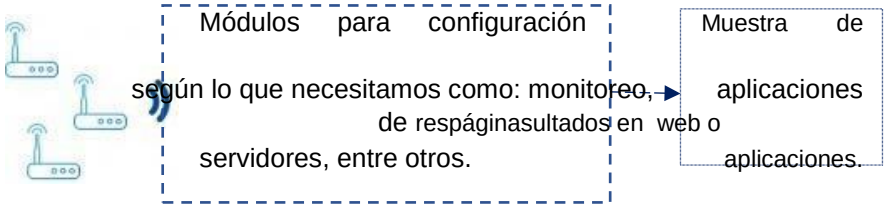

Figura 1.1 Funcionamiento de LoraWAN [3]

Como se muestra en la Figura 1.1, al usar LoraWAN podemos hacer la interconexión de dispositivos o de cosas inteligentes sin hacer instalaciones complejas, dando más libertad para desarrollo a la persona o entidad que lo esté usando el dispositivo y al final pueda visualizar los resultados.[3]

1.4.3. SENSOR DE HUMEDAD Y TEMPERATURA - DHT11

El DHT11 es un sensor pequeño que permite medir temperatura y humedad del área donde se lo ubica, es económico y fácil de usar. [4]

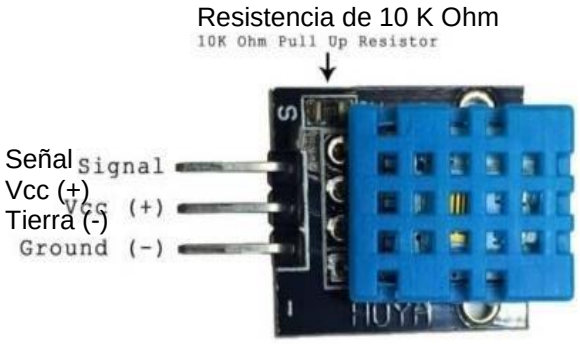

Figura 1.2 DHT11 – Sensor de humedad y temperatura.[4]

El sensor que encontré en el mercado es el que se muestra en la figura 1.2., éste está sobrepuesto en una minúscula placa que saca 3 pines para ser usados de la siguiente forma: 1 para energizar, 1 para conectarlo a tierra y 1 para el paso de la señal. [4]

Características del DHT11:

Dentro del datasheet de este sensor se encuentran muchas características, a continuación, se va a mencionar las que necesitamos para el desarrollo de este proyecto:

- Voltaje de operación: 3,3 V a 5V
- Corriente 2,5 mA
- Transmite hacia el controlador el resultado de señal digital
- El rango de temperatura que mide es de 0°C a 50°C con una precisión de +/- 2°C
- El rango de humedad que mide es de 20% a 90% HR, con una precisión de +/- 5%.

1.4.4. LORA SHIELD V1.4

El Lora Shield V1.4, es una placa con circuitos integrados como se muestra en la figura 1.3 y en la figura 1.4, de la empresa Dragino, basado en una biblioteca de código abierto que permite enviar datos y alcanzar en tramos largos y a buena velocidad. Usa la técnica de modulación SX127x LoRa®.[5]

Figura 1.3 LoRa Shield V 1.4 – Vista frontal

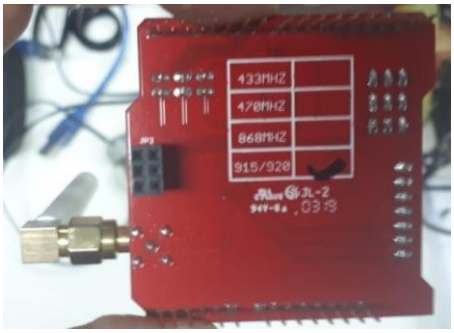

Figura 1.4 LoRa Shield V1.4 – Vista posterior

Características Principales de LoRa Shield V 1.4:

- Es compatible con una placa Arduino con voltaje de operación de entrada y salida, que va desde los 3.3v a los 5v. En este proyecto se trabajará con una placa Mega.
- La frecuencia escogida para este proyecto es: 915MHz
- Consume poca energía en su funcionamiento
- Tiene una antena externa, que transmite vía WiFi

1.4.5. LG01-P

El LG01-P conocido también como LoRa Gateway, es un dispositivo de Dragino, y tiene la forma que se muestra en la figura 1.5. Se programa en código abierto, el usuario decidirá si ejecuta el código que viene cargado en el sistema o lo modifica dependiendo la función que quiere obtener. Se conecta con una LoRa Shield de forma inalámbrica, como podemos apreciarlo en la figura 1.6, lo que permite enviar datos en distancias largas, no genera interferencias. [6]

Figura 1.5 LG01-P [6]

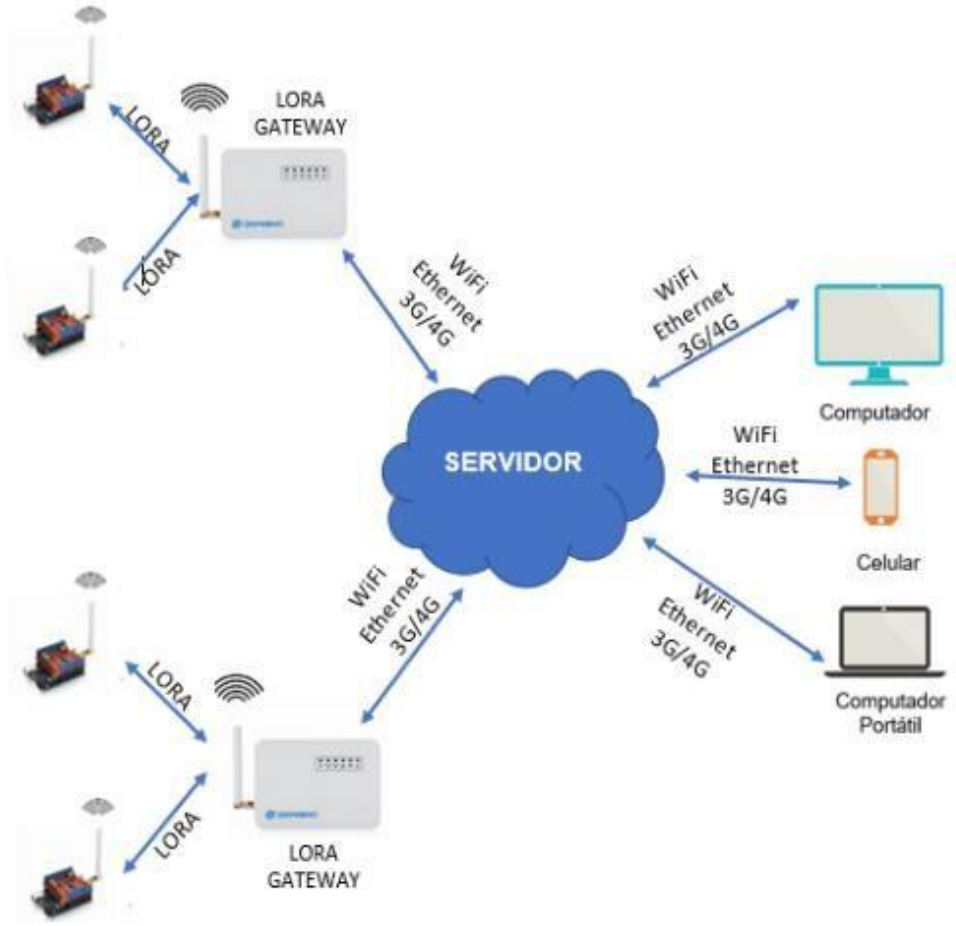

Figura 1.6 Conexión ejemplar usando LoRa Shield y LoRa Gateway[6]

1.4.6. ARDUINO MEGA 2560

El Arduino mega 2560 es una placa como se muestra en la figura 1.7, que usa el controlador ATmega2560. Es compatible con la mayoría de placas de expansión como LoRa Shield. Esta versión posee mayor capacidad de procesamiento, memoria y más líneas de expansión comparada a otros Arduinos.[7]

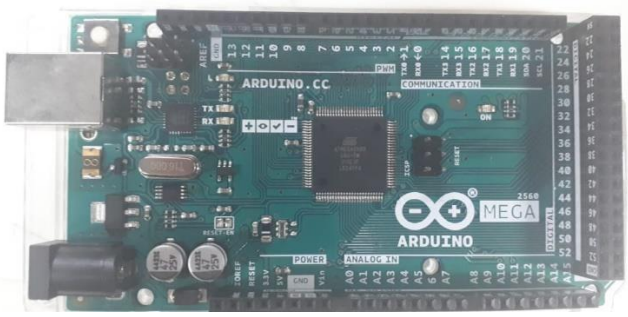

Figura 1.7 Arduino Mega 2560.[7]

1.4.7. RASPBERRY PI 3

La Raspberry pi 3 como se muestra en la figura 1.8, es una placa que tiene embebidos muchos componentes que lo hacen funcionar como un ordenador, para usarla tenemos que contar con un monitor, un teclado, un mouse y un cable de conexión HDMI. Es de tercera generación, salió a la venta en febrero de 2016. El fabricante indicó que se estará vigente al menos hasta enero de 2026.[8]

Figura 1.8 Raspberry Pi 3 [8]

1.4.8. DOCKER

Sirve para crear código basado en contenedores, haciendo que la programación sea más fácil y pueda ser compartida. Al hablar de contenedores, podemos imaginar a los buques, un solo buque lleva varios contenedores y cada contenedor lleva en su interior diferentes productos. Este es el principio que se usa en los contenedores de software de Docker,

es un programa donde podemos ejecutar varias aplicaciones en tiempo real que realicen diferentes funciones, sin que esto afecte la configuración y sistema operativo que esté usando. [10]

Una ventaja importante de Docker es que se puede crear una aplicación en una versión diferente del programa y ejecutarlo en una versión más actual sin que afecte la compilación de la aplicación, ya que internamente tiene todo lo que necesita para configurarse, como se muestra en la figura 1.9. [10]

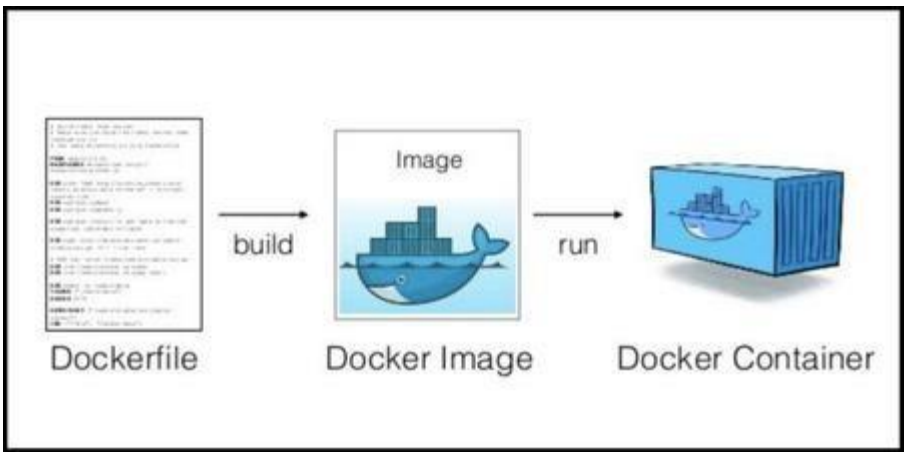

Figura 1.9 Funcionamiento interno de los contenedores [10]

Comparación de contenedores vs. Virtualización.

En la virtualización, tenemos varias máquinas que trabajan a la par, pero cada una debe tener su propio sistema operativo lo que hace que consuman más recursos en el sistema operativo anfitrión. [10]

Los contenedores también usarán el sistema operativo anfitrión, pero no necesitan instalación de cada sistema operativo que vaya a usar, ya que todo lo que necesita para su funcionamiento está integrado en ellos. Son más ligeros, ya que usa el Kernel del anfitrión. En la figura 1.10, se hace estimación del uso de recursos que se usan en una máquina virtual vs los recursos que se usan en un contenedor. [10]

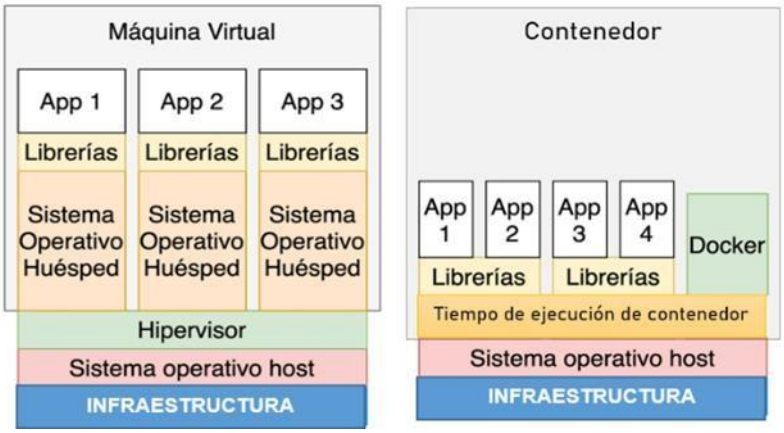

Figura 1.10 Comparación del uso de recursos entre una Máquina virtual y un contenedor [10][17]

Ventajas de un contenedor Docker vs. máquinas virtuales.

- El contenedor puede usar megabytes, mientras la misma aplicación configurada en una máquina virtual puede usar gigabytes.
- Cada aplicación de Docker tiene su propio contenedor aislado, mientras que en las máquinas virtuales tiene varias aplicaciones para su funcionamiento.

1.4.9. NUC7JYB

Los NUC son pequeñas computadoras, que vienen de fábrica solo con la tarjeta madre, el procesador y la caja, como se muestra en la figura 1.11. Adicionalmente hay que comprar tarjetas de memoria RAM y disco duro, se puede continuar ensamblando según el uso que deseemos darle. [14]

Figura 1.11 NUC7JYB [18]

CAPÍTULO 2

DISEÑO DEL PROTOTIPO DE MEDICIÓN

2.1. CONFIGURACIÓN DEL PROTOTIPO Y DESARROLLO

Para el desarrollo de este prototipo se investigó que equipos y tarjetas se podían usar, con la finalidad de lograr los objetivos trazados en la propuesta.

La configuración de este prototipo se realizó en 3 partes, como se detalla a continuación:

- **Parte 1:** Conexión y programación del sensor DHT11 hacia el Arduino MEGA 2560 con el LoRa Shield.

  Al inicio tenía previsto usar el sensor DHT22, pero no se encontró en el mercado ecuatoriano. En las electrónicas donde se consultó, recomendaron el sensor DHT11 que tiene características similares al sensor DHT22.

  A continuación, se muestra la tabla 1 con un breve cuadro comparativo:

	DHT11	DHT22
Alimentación	De 3 a 5V	De 3 a 5V
Corriente máxima	2.5mA máx.	2.5mA máx.
Rango de Humedad	20-80% / 5%	0-100% / 2-5%
Rango de Temperatura	De 0°C a 50°C / ± 2°C	De -40°C a 80°C / ± 0.5°C
Tasa de muestreo	1 Hz (Leyendo cada segundo)	0.5 Hz (Leyendo cada 2 segundos)
Tamaño	15.5mm x 12mm x 5.5mm	15.1mm x 25mm x 7.7mm
Ventajas	Bajo Costo	Más preciso

Tabla 1: Comparación entre DHT11 y DHT22[9]

Una vez escogido el sensor, se revisa puertos y conectores del Arduino Mega 2560 y LoRa Shield, y se realiza la conexión física como se muestra en la figura 2.1.

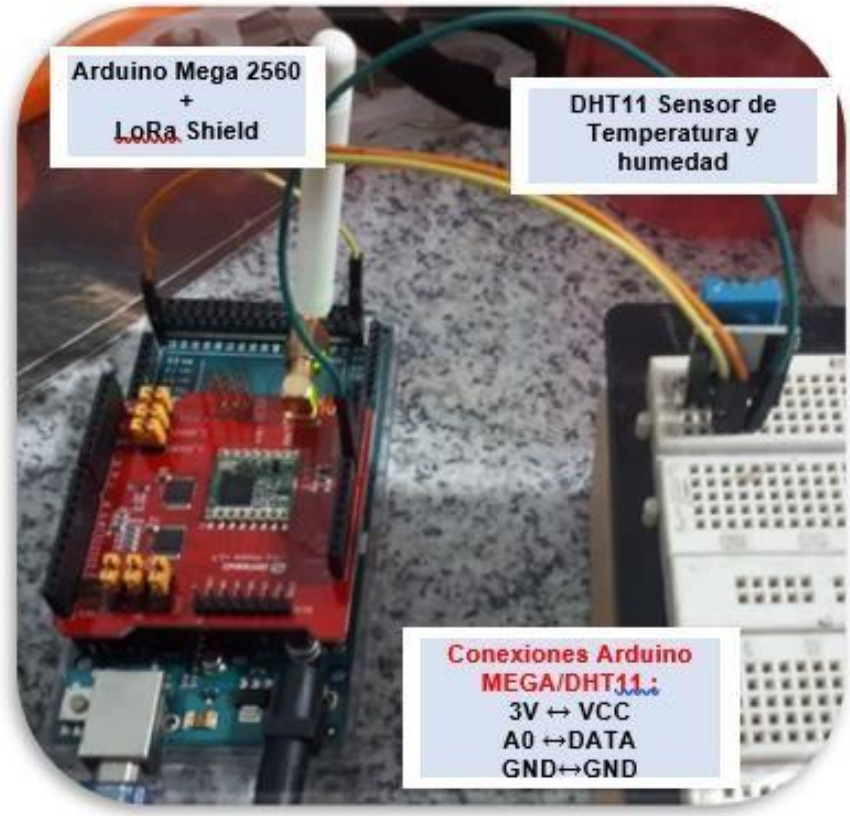

Figura 2.1 Primera parte ensamblada

Como requisito tener instalado en la computadora el programa Arduino IDE.

Se conecta el Arduino Mega 2560 por medio del puerto USB a la computadora, se abre en el Arduino IDE el archivo modelo llamado LoRa_Simple_Client_DHT11, desde la librería de LoRa Shield.

Una vez está abierto el programa Arduino IDE, dar clic en archivo -> ejemplos en el siguiente menú se seleccionada Dragino y el resto de ruta es como se muestra en la figura 2.2 hasta encontrar ejemplo precargado:

LoRa_Simple_Client., el mismo que se modificará de acuerdo a las necesidades de este proyecto.

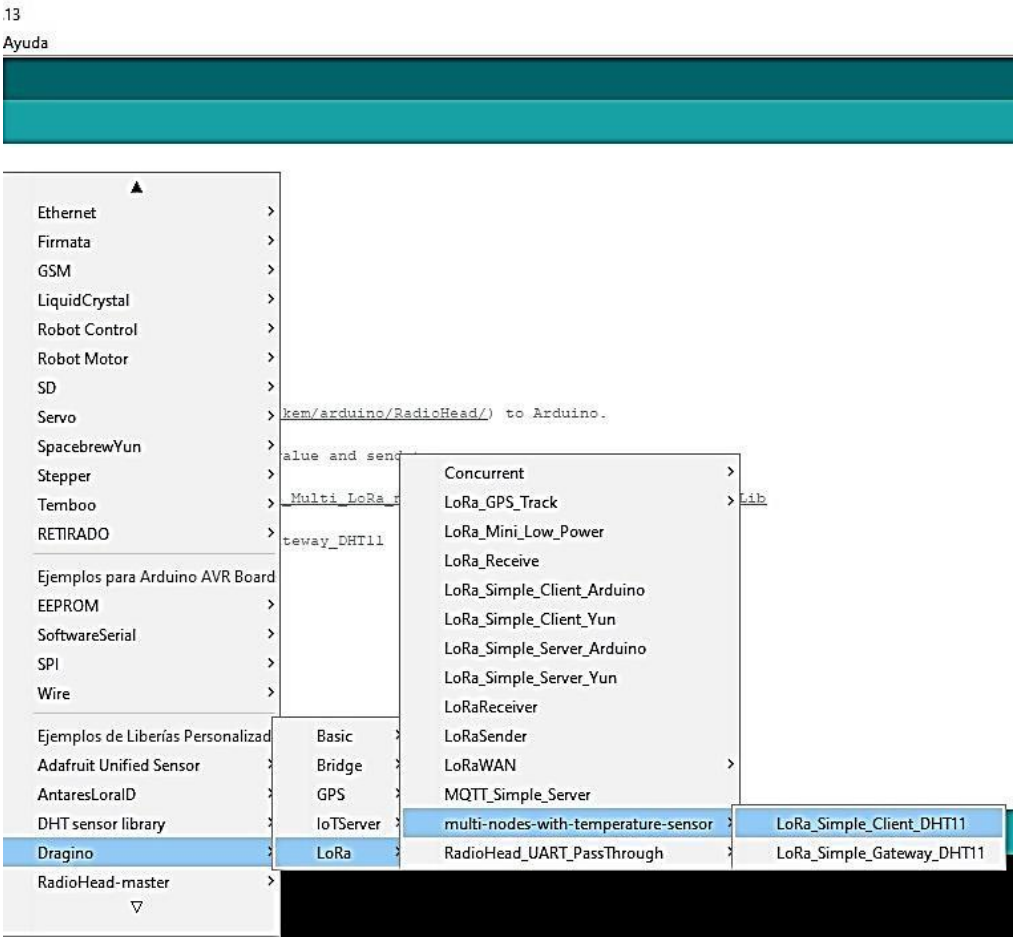

Figura 2.2 Código ejemplo LoRa_Simple_Client

Al abrir el programa se modificará la frecuencia de operación a 920MHz para que exista conectividad con la LoRa Shield, cuya frecuencia de operación es de 915/920 MHz.

En la figura 2.3. se muestra el código que se compilará en el programa Arduino IDE y que se carga en la placa Arduino Mega 2560.

```
#include <SPI.h>
#include <RH_RF95.h>
#include <String.h>

RH_RF95 rf95;
float frequency = 920.0; // Change the frequency here.

#define dht_dpin A0 // Use A0 pin to connect the data line of DHT11
byte bGlobalErr;
char dht_dat[5];
```

Figura 2.3 Código para captar datos con el DHT11

En la pantalla de ejecución del pro grama Arduino IDE, se mostrarán las métricas de temperatura y humedad obtenidas por el DHT11, procesadas en el Arduino Mega 2560 y están listas para la transmisión, se muestra los valores captados en la figura 2.4.

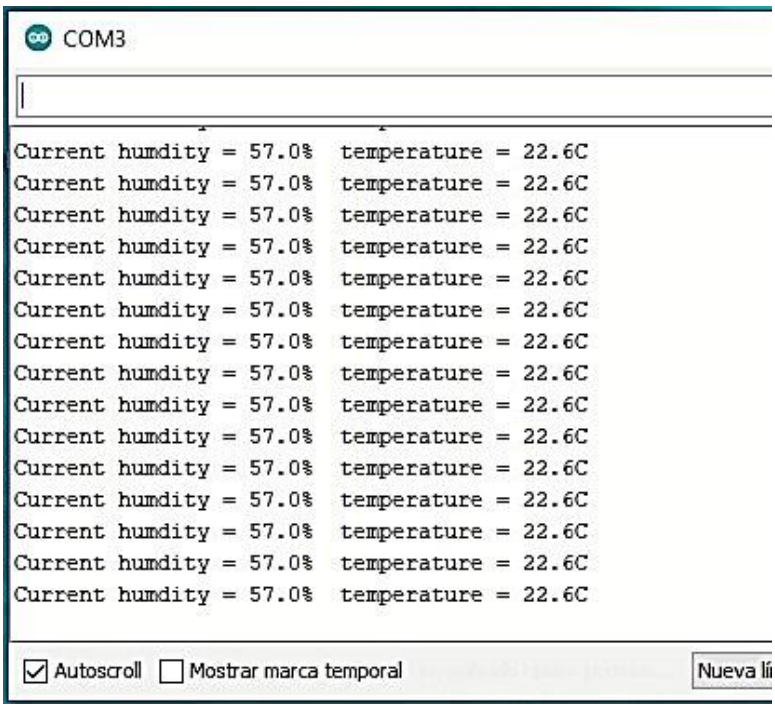

Figura 2.4 Muestra de valores captados por el sensor.

- **Parte 2:** Conexión entre la parte 1 y el LoRa Gateway.

  Se conecta el LoRa Gateway al computador para hacer la compilación y carga del programa. Se abre el programa Arduino IDE, dar clic en archivo -> ejemplos en el siguiente menú se seleccionada Dragino y el resto de ruta es

como se muestra en la figura 2.5 hasta encontrar ejemplo el código ejemplo LoRa_Simple_Gateway.

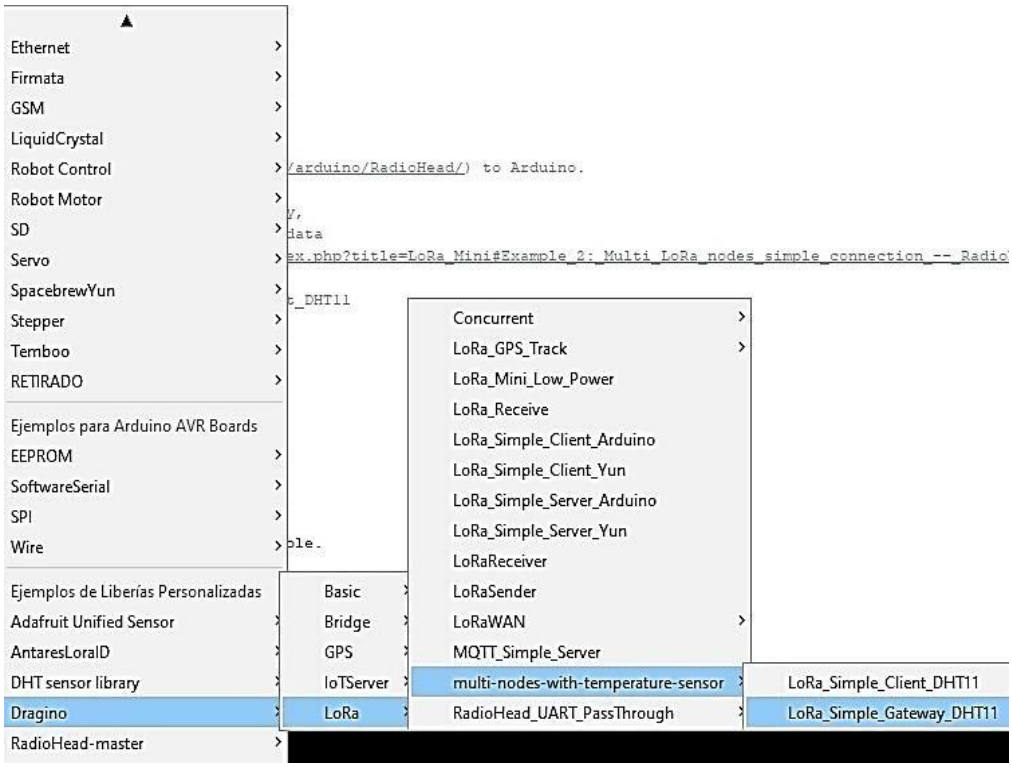

Figura 2.5 Código ejemplo LoRa_Simple_Gateway

Se modifica la frecuencia de operación del código precargado, esto se hace para que exista conexión entre el LoRa Shield y el LoRa Gateway, y así establece la conexión de la segunda parte.

- **<u>Parte 3:</u> Conexión entre la parte 2 y el Raspberry pi.**

  Al igual que en la parte 2, se configura el código del programa LoRa_Simple_Gateway, para que a la salida de datos del LoRa Gateway no haya conflictos y se logre hacer la transmisión de datos, vía WiFi hacia la Raspberry Pi.

  En la figura 2.6, apreciamos la conexión física de los elementos ya programados, el Dragino está listo para crear la base de las métricas y conectarse a la Raspberry Pi.

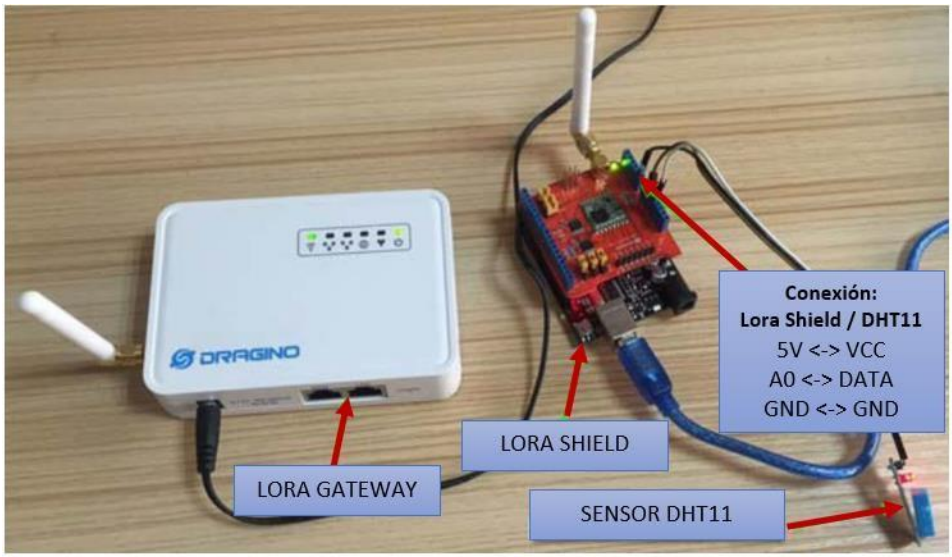

Figura 2.6 Configuración física del prototipo, sin conexión al Raspberry Pi

Para establecer la comunicación entre el LoRa Gateway y el Raspberry Pi, se debe modificar la siguiente línea de comando, que se encuentra en el archivo LoRa_Simple_Gateway:

```
client.get("http://10.130.1.10:4000/sensores?id=1&temp=" + (String) b + "&hum=" + (String) a);
```

En el código se realizó una conversión de tipo de datos, obligando a que el dato que llegue del LoRa Gateway salga hacia el Raspberry pi como un dato tipo string(cadena), que será usado en la base de datos a diseñar.

En el código preconfigurado se hace mención al servidor Thingspeak del propio Dragino, pero esto no es lo se quería. Por eso se configura la dirección IP 10.130.1.10 en la interfaz de red wifi del Raspberry Pi, para que éste pueda recibir los datos obtenidos de base sensores y comenzarlos a interpretar.

A continuación, se va mostrar el lugar donde se instalaron los dispositivos, se debe mencionar que solo se tuvo acceso a un rack designado por jefatura.

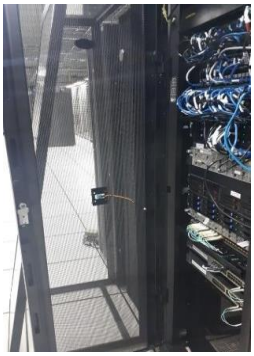

Figura 2.7Sensor ubicado en el rack vista panorámica.

Figura 2.8 Vista del pasillo frío del rack

En la figura 2.7 se muestra el sitio donde se colocó el prototipo, se tomó la foto con la puerta abierta del rack. En la figura 2.8 se muestra al rack con la puerta cerrada, se puede apreciar algo parecido a una caja que se denomina jaula y en el piso se encuentran las baldosas por donde el aire ingresa, a eso se le llama pasillo frío.

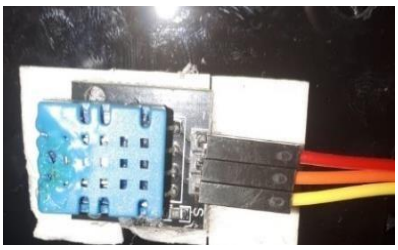

Figura 2.10 Sensor sobrepuesto en base acrílico por estética.

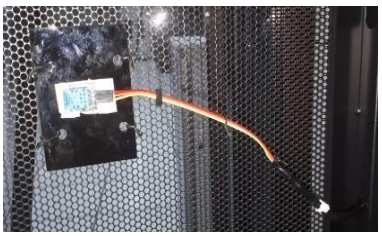

Figura 2.9 Sensor instalado en la puerta del rack.

En la figura 2.9 se muestra el sensor DHT11 acoplado a una pequeña placa de acrílico solo por estética. En la figura 2.10 se muestra el sensor ya instalado en la puerta del rack.

Figura 2.11 Ubicación en sitio del LoRa Shield y Arduino Mega 2560

En la figura 2.11, se muestra el acoplamiento del LoRa Shield con el Arduino Mega 2560, que está conectado en las borneras con el sensor.

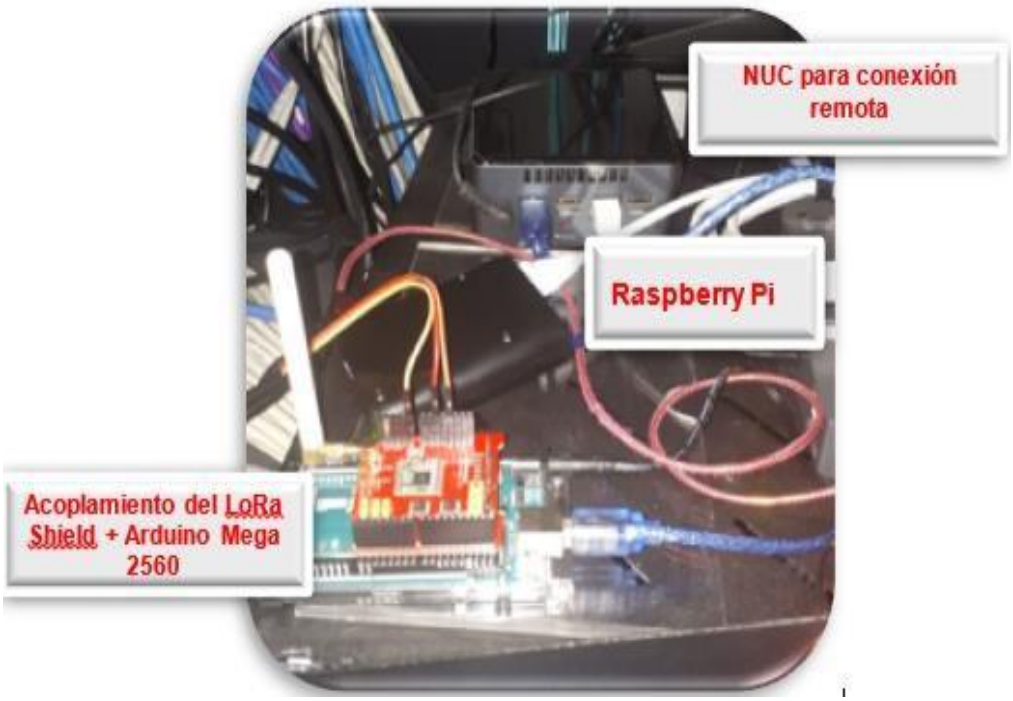

Figura 2.12 Conexión del prototipo

En la figura 2.12 se muestra la conexión del prototipo se la hizo dentro del rack porque no había un espacio físico donde ubicarlos, ahí están ubicados en la parte de afuera el sensor DTH11 como antes se ha indicado, el LoRa Shield acoplado al Arduino Mega 2560, el Raspberry Pi 3 y una NUC que se

usará para visualizar los datos. Como se pueden dar cuenta el LoRa Gateway no está en la foto, ya que está ubicada a una distancia de 12 metros fuera del rack.

2.2. DISEÑO DE LA BASE DE DATOS

Para el diseño de la base de datos, lo primero que se debe hacer es instalar Docker en la Raspberry Pi y configurar los contenedores.

Un contenedor puede tener muchas bases de datos y esas bases de datos a su vez pueden tener más bases de datos, que guarden diferentes mediciones. En este caso, solo vamos a usar una base de datos, la que se llamará sensores.

Los contenedores que se usarán serán: influxdb y chronograf

- Contenedor influxdb: es una base de datos documental, cuyo diseño no es difícil en comparación con otras bases de datos de series de tiempo.[12] A continuación, se muestra el código para la programación del contenedor de la base Sensores usando influxdb:

```
services:
influxdb:
   image: influxdb:1.8
ports:      -
'8086:8086'    restart:
always    volumes:
- influxdb-storage:/var/lib/influxdb
  environment:
- INFLUXDB_DB=sensores
- INFLUXDB_ADMIN_USER=admin
- INFLUXDB_ADMIN_PASSWORD=admin
- INFLUXDB_USER=user
- INFLUXDB_USER_PASSWORD=user
```

```
 server:
build:
    context: ./server
ports:      -
'4000:4000'
restart: always
depends_on:
- influxdb
```

- Contenedor Chronograf: es una herramienta gráfica que se usa junto a influxdb, para crear reglas de alerta y automatización de procesos. [11] Es una API sencilla de usar, comparada a la línea de comandos. Para este diseño usaremos Chronograf, solo al inicio, para establecer las políticas de retención a la base de datos.

```
 chronograf:
   image: chronograf:1.8
ports:      - 8888:8888
restart: always
environment:
- INFLUXDB_URL=http://influxdb:8086
- INFLUXDB_USERNAME=admin      -
 INFLUXDB_PASSWORD=admin    volumes:
- chronograf-storage:/var/lib/chronograf
 depends_on:
- influxdb
```

Para que los contenedores funcionen, se debe cargar en el Raspberry Pi 3, una función controladora que recepta la data del LoRa Gateway y la almacena

en la base de datos sensores usando el influxdb, en esta función se establece los días que permanecerán guardados los datos, antes de que se sobrescribirán.

2.3. CONFIGURACIÓN DEL VISOR WEB

Para el visor web también usaremos contenedores de Docker que estarán configurados en la Raspberry Pi 3.

Lo que se espera ver en las gráficas, es los datos de la base sensores.

Para el visor web, usaremos Grafana, que permite visualizar métricas, almacenadas en cualquier base de datos [13]. La configuración de este contenedor se la muestra a continuación:

```
grafana:
  image: grafana/grafana:7.0.0
ports:    - 3000:3000
restart: always    volumes:
- grafana-storage:/var/lib/grafana    depends_on:
- influxdb
```

2.4. EJECUCIÓN DE PRUEBAS

En esta sección mostraremos las pruebas de la base de datos y del visor web, realizadas a través de la consola de la NUC, denominada powershell de windows y de ahí se conecta por SSH al Raspberry PI 3.

Estas pruebas también las puedo hacer por líneas de comandos directamente en el Raspberry Pi 3.

Por motivos de presentación, se conectó la NUC al Raspberry Pi 3, para tomar pruebas de vía remota.

2.4.1. PRUEBAS BASES DE DATOS

Se mostrarán que se creó la base de datos, a través de consultas realizadas usando la consola de Docker, para ver información de la base de datos

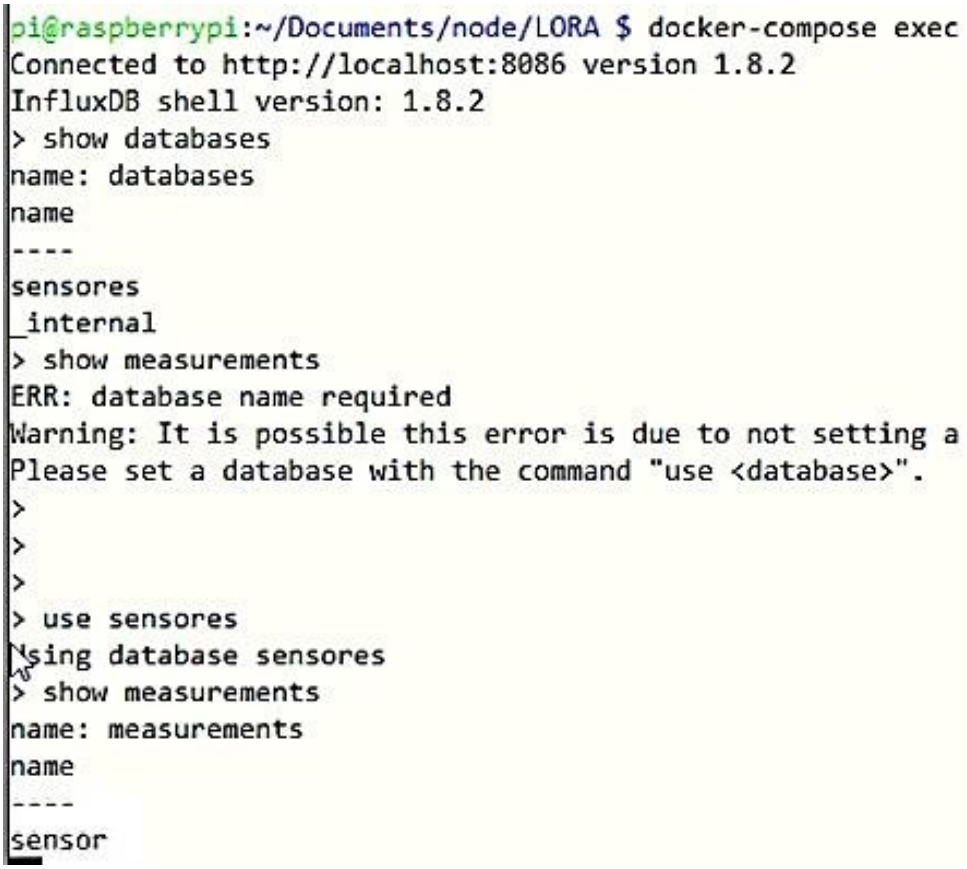

Figura 2.13 Nombre de la base de datos

En la figura 2.13, se puede apreciar que usando los comandos:

- show databases: se puede ver el nombre de la base de datos
- show measurements: para ver las mediciones, y donde se guarda la data

```
> select * from tres_dias.sensor order by time desc limit 20
name: sensor
time                           codigo fila humedad id rack temperatura
----                           ------ ---- ------- -- ---- -----------
2020-09-12T17:50:58.948242854Z AXT5   8    57      1  5    22
2020-09-12T17:50:54.850413845Z AXT5   8    57      1  5    22
2020-09-12T17:50:50.783390982Z AXT5   8    57      1  5    22
2020-09-12T17:50:47.069218223Z AXT5   8    57      1  5    22
2020-09-12T17:50:42.972899006Z AXT5   8    57      1  5    22
2020-09-12T17:50:38.876655518Z AXT5   8    57      1  5    22
2020-09-12T17:50:34.781214946Z AXT5   8    57      1  5    22
2020-09-12T17:50:30.68409099Z  AXT5   8    57      1  5    22
2020-09-12T17:50:26.589527085Z AXT5   8    57      1  5    22
2020-09-12T17:50:22.697613961Z AXT5   8    57      1  5    22
2020-09-12T17:50:18.601908546Z AXT5   8    57      1  5    22
2020-09-12T17:50:14.520589954Z AXT5   8    57      1  5    22
2020-09-12T17:50:10.408360633Z AXT5   8    57      1  5    22
2020-09-12T17:50:06.518756884Z AXT5   8    57      1  5    22
2020-09-12T17:50:02.423121625Z AXT5   8    57      1  5    22
2020-09-12T17:49:58.327516627Z AXT5   8    57      1  5    22
2020-09-12T17:49:54.435084389Z AXT5   8    57      1  5    22
2020-09-12T17:49:50.20703439Z  AXT5   8    57      1  5    22
2020-09-12T17:49:46.652359704Z AXT5   8    57      1  5    22
2020-09-12T17:49:42.147535591Z AXT5   8    57      1  5    22
>
```

Figura 2.14 Datos almacenados

En la figura 2.14, se puede apreciar que usando el comando:

- select * from tres_dias.sensor order by time desc limit 20: se observa el resultado de los últimos 20 registros guardados en la base de datos.

2.4.2. PRUEBAS DEL USO DE DOCKER

- En la figura 2.15, se puede apreciar todas las imágenes de Docker creados en Raspberry Pi 3, usando el siguiente comando: docker images

```
pi@raspberrypi:~/Documents/node/LORA $ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
lora_server         latest              951ac3ea3ed0        9 days ago          748MB
<none>              <none>              53aed07901d2        9 days ago          748MB
chronograf          1.8                 b445e5d5f0d0        11 days ago         160MB
influxdb            1.8                 826eb2cce673        12 days ago         261MB
node                10                  b697c995167b        12 days ago         744MB
grafana/grafana     7.0.0               f58f4774cccb        4 months ago        127MB
hello-world         latest              851163c78e4a        8 months ago        4.85kB
```

Figura 2.15 Imágenes Docker

• Para conocer el número de contenedores que se están usando en este proyecto, se escribe el comando: docker-compose up -d

```
pi@raspberrypi:~/Documents/node/LORA $ docker-compose up -d
lora_influxdb_1 is up-to-date
lora_server_1 is up-to-date
lora_chronograf_1 is up-to-date
lora_grafana_1 is up-to-date
```

Figura 2.16 Contenedores Docker

2.4.3. PRUEBAS DEL VISOR WEB

En la figura 2.17, se muestra la interfaz gráfica del Grafana, a esta se accede desde el NUC, por el navegador web con la dirección IP con su puerto de salida: 172.30.9.50:3000

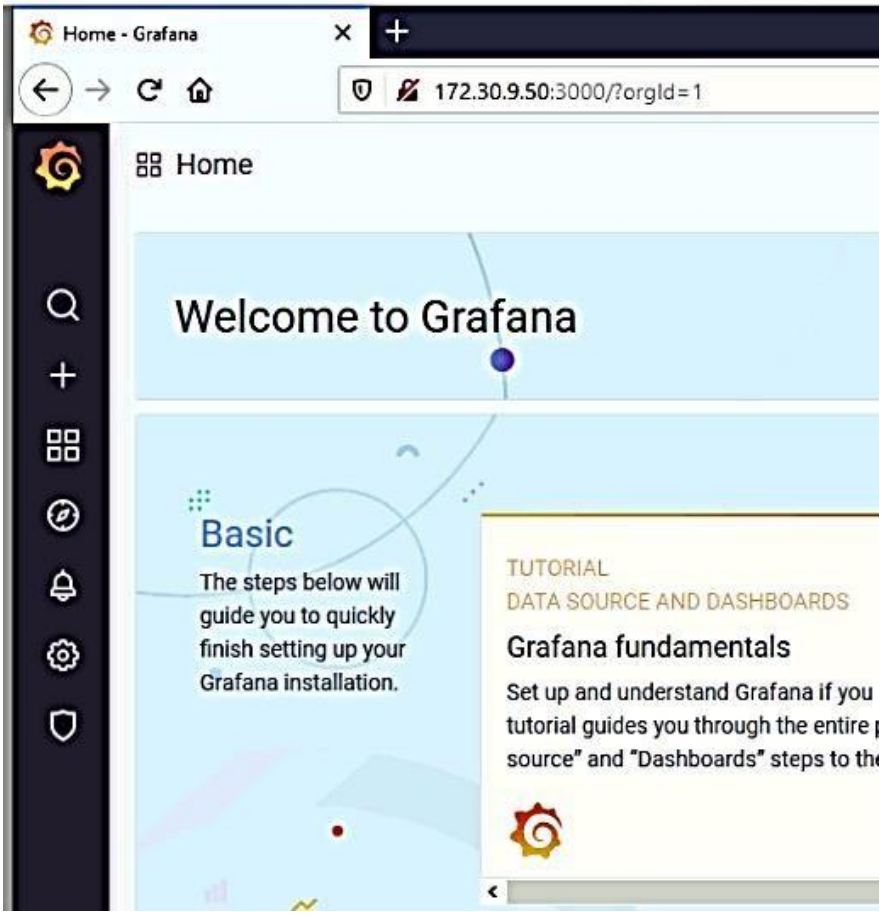

Figura 2.17 Interfaz gráfica del Grafana

La dirección IP: 172.30.9.50, mencionada en el párrafo anterior está configurada en la tarjeta de red del Raspberry Pi 3. El NUC está configurado con la dirección IP: 172.30.9.1. Lo que indica que ambos equipos están en la misma Red. Cualquier equipo que esté en la misma red y posea un navegador web, puede acceder a Grafana y ver el monitoreo del sensor.

En la figura 2.18, se indica como acceder al visor web para observar los resultados medidos. Dar clic en dashboard y clic en manage.

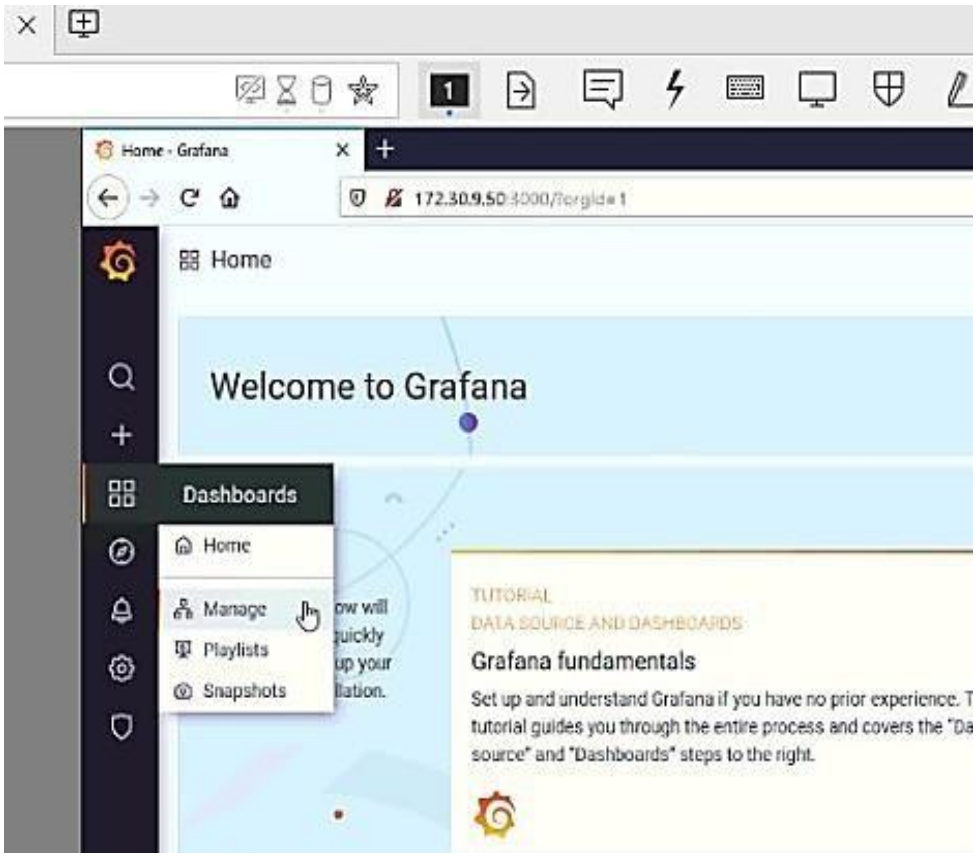

Figura 2.18 Acceso al Visor 1/2

La siguiente ventana es la que se muestra en la figura 2.19, ahí debemos dar clic en la palabra REGISTRO.

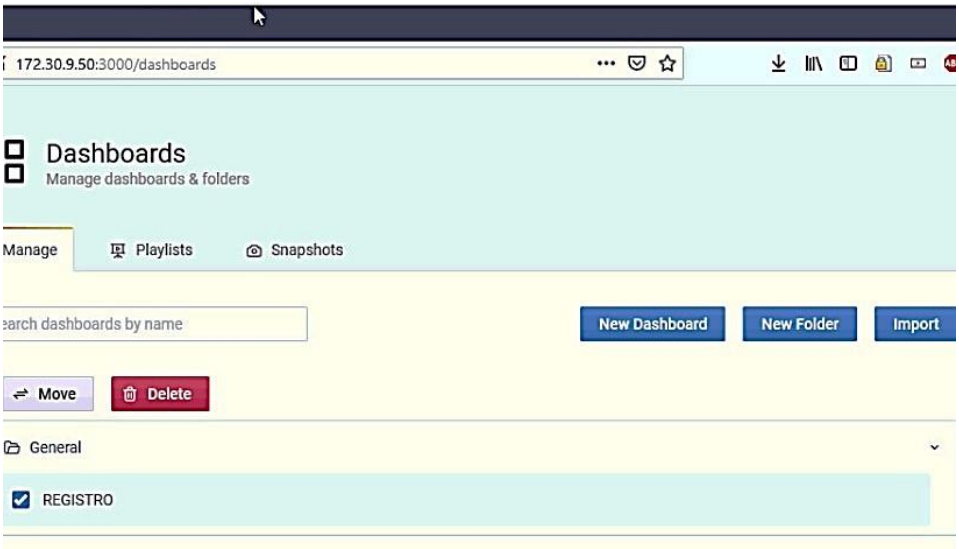

Figura 2.19 Acceso al Visor 2/2

Al dar doble clic en el botón registro se mostrará la figura 2.20, en ella podemos observar las gráficas de temperatura y humedad que se encuentra midiendo el sensor.

Figura 2.20 Gráficas de temperatura y humedad

La figura 2.21, muestra la configuración en Grafana, para que los datos obtenidos de la base sensores se muestren tal cual en la figura 2.20.

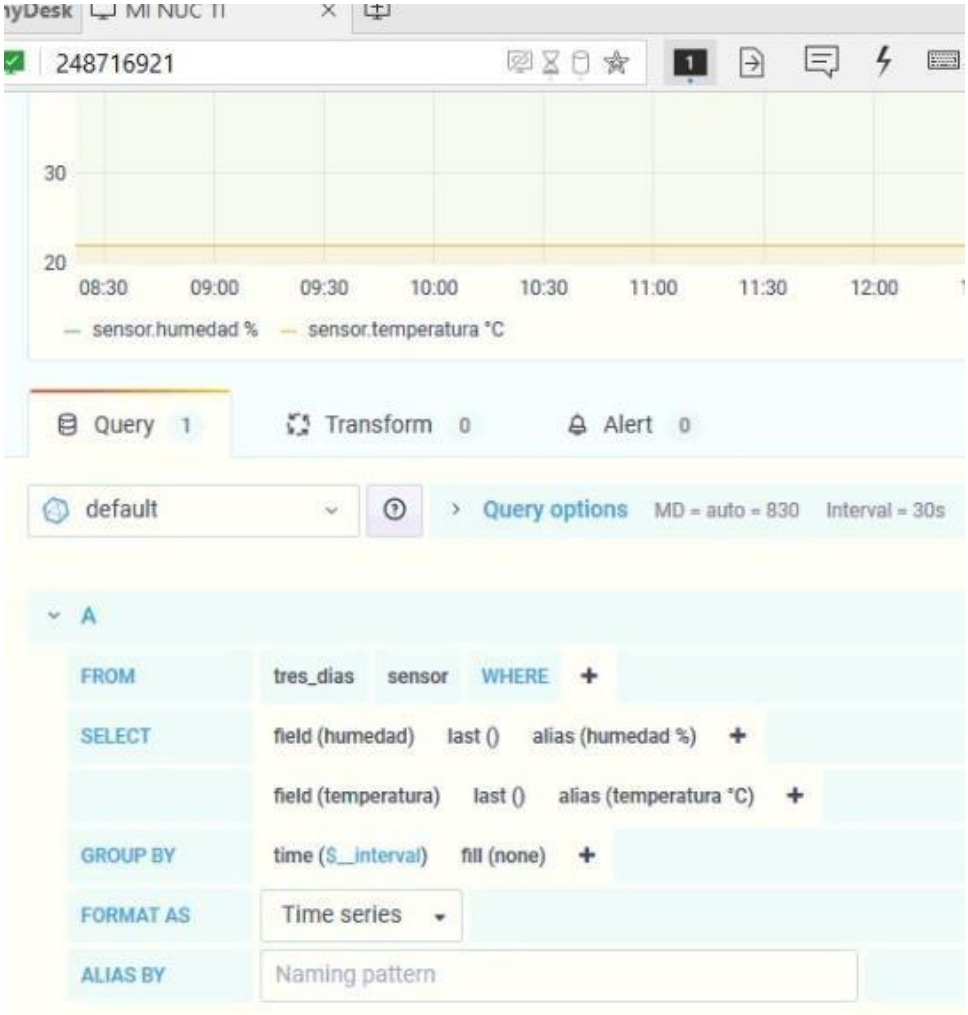

Figura 2.21 Configuración para muestreo en Grafana

CAPÍTULO 3

ANÁLISIS DE RESULTADOS

Estas pruebas permiten verificar si el prototipo cumple o no, con la solución propuesta en este proyecto.

3.1. ANÁLISIS DE RESULTADOS DE LAS PRUEBAS

Los resultados obtenidos en los últimos días que se hicieron las pruebas fueron constantes, el Grafana siempre mostró una temperatura de constante de 22°C con una humedad de 57% como se muestra en la figura 3.1.

Figura 3.1 Gráfico de Humedad y Temperatura

Se alejó más el LoRa Gateway del LoRa Shield, para saber si había variación en las mediciones, pero el resultado no cambió.

Se probó el sensor DTH11 fuera de los racks, para validar que variaba la temperatura y si se confirmó que el sensor funcionaba correctamente.

No sé pudo realizar pruebas ubicando el sensor en otros racks.

Con respecto a los resultados obtenidos por sensor DTH11, las imágenes de Grafana y tomando en cuenta la presión del sensor de humedad y temperatura, se puede indicar que la temperatura y humedad están dentro del rango que establece la norma TIA/EIA-942 en la que se recomienda que el rango aceptable de temperatura esté entre 20 °C y 25 °C. [15]

3.2. ANÁLISIS COMPARATIVO DE LOS RESULTADOS

Se va a comparar los resultados obtenidos del prototipo vs la información configurada en las UMA's, las mismas que son los equipos de enfriamiento del cuarto de racks.

En el cuarto de racks existen varias UMA's para análisis se debe escoger la más cercana al rack en el que se encuentra el sensor.

- **HUMEDAD:**

  Según la norma (TIA/EIA 942) el rango de humedad adecuado en un centro de datos es de 40% a 55%, con el fin de evitar descargas eléctricas y de condensación. [16]

  Si tomamos en cuenta que la precisión del sensor es más/menos 5%, entonces se puede indicar que la HR se encuentra en el rango adecuado para su funcionamiento. Siendo 57% el valor de HR que se obtiene en el Grafana, y 47% el valor que indica la UMA5, como se muestra en la figura 3.1. y en la figura 3.2, respectivamente.

Figura 3.2 UMA 5 - MÉTRICA HUMEDAD

Considerar: si la HR es muy alta puede causar problemas de estática por el ambiento seco y si es muy baja puede causar condensación y generar partículas de agua [16].

- **TEMPERATURA:**

  Para las comparaciones se considerará los datos de la figura 3.3., en la que existen 3 parámetros de temperatura.

Figura 3.3 UMA7 - TEMPERATURA

La temperatura del aire suministrado es: 15.3 °C. Este aire sale desde la UMA y viaja por los ductos de aire hasta llegar al ducto final, que está muy cercano al rack en que se tomó datos.

1. La temperatura que retorna es: 23.0°C, se mantiene en el umbral de valores óptimos.
2. La temperatura en que se debe mantener es 22.5°C.

La comparación se realizará con la temperatura del numeral 3, que es la que se espera encontrar en los racks.

En la figura 3.1., la temperatura está en 22°C si tomamos en cuenta el rango de precisión +/- 2°C y lo comparamos con la temperatura del numeral 3. Se puede indicar que la temperatura es óptima para el buen funcionamiento de los equipos.

CAPÍTULO 4

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

- Se logró el objetivo principal del proyecto, que era medir la temperatura y humedad de un rack en un centro de datos, usando el protocolo LoRa Shield con un sensor.
- La variación de la temperatura, entre el equipo de enfriamiento externo y la medida de climatización que capta el sensor, es diferente en más/menos 2 grados por la precisión del sensor. Si la temperatura es superior a 25%, deben tomarse acciones inmediatas porque es peligroso para los equipos.
- La variación de la temperatura, es diferente incluido los más/menos 5% de precisión. Pero, en ambos casos está dentro de los parámetros recomendados.
- Se configuró la base de datos para que almacenara la data por 3 días, ya que solo se necesita esta información para monitoreo, después de eso los datos se sobrescriben.
- Los gráficos que se muestran como resultado de la base de datos pueden observarse en Grafana individualmente, y dependerá de la hora y fecha que se escoja.

RECOMENDACIONES

- Verificar la frecuencia de trabajo en los dispositivos transmisores y receptores a usarse, para que no haya problemas de comunicación.
- Agregar más nodos en diferentes racks para obtener resultados más precisos del cuarto de racks, para este proyecto solo se tuvo acceso a un rack.
- En futuros trabajos se podría agregar otro sensor con mayor precisión con la finalidad de verificar la veracidad de los datos obtenidos.

BIBLIOGRAFÍA

[1] "LoRaWAN", Medium, 2020. [Online]. Disponible: https://medium.com/pruebas-delaboratorio-de-la-modulaci%C3%B3n-lora/lorawan-d00f48384160. [Acceso: 20 - Agos - 2020].

[2] "Internet de las cosas; Qué es, ejemplos y sus ventajas y desventajas", Noticias eficiencia energética y arquitectura | OVACEN, 2020. [Online]. Disponible: https://ovacen.com/internet-de-las-cosas/. [Acceso: 31- Agos- 2020].

[3] "Tecnología LoRA y LoRAWAN - Catsensors", Catsensors.com, 2020. [Online]. Disponible: https://www.catsensors.com/es/lorawan/tecnologia-lora-y-lorawan. [Acceso: 31- Agos- 2020].

[4] "DHT11 - Sensor de humedad y temperatura - Ultra-lab", Ultra-lab, 2020. [Online]. Disponible: http://ultra-lab.net/producto/dht11-sensor-de-humedad-y-temperatura/. [Acceso: 31- Agos - 2020]

[5] "Arduino Shield featuring LoRa® technology", Dragino.com, 2020. [Online]. Disponible: http://www.dragino.com/products/lora/item/102-lora-shield.html. [Acceso: 06- Sep- 2020]

[6] “LG01-P IoT Gateway featuring LoRa® technology”, Dragino.com, 2019. [Online]. Disponible: http://www.dragino.com/products/lora-lorawan-gateway/item/117-lg01p.html [Acceso: 06 – Sep - 2020]

[7] “Arduino Mega 2560 Características, Especificaciones | Proyecto Arduino.” Proyecto Arduino, 2020. [Online]. Disponible: proyectoarduino.com/arduinomega-2560/.[Acceso: 06 – Sep – 2020]

[8] “Buy a Raspberry Pi 3 Model B – Raspberry Pi.”, Raspberry. [Online]. Disponible: raspberrypi.org/products/raspberry-pi-3-model-b/ [Acesso: 09 – Sep - 2020]

[9] “DHT11 Y DHT22 Sensor De Temperatura Y Humedad”. [Online]. Disponible: https://saber.patagoniatec.com/2014/06/dht22-dht11-sensor-de-tempertatura-yhumedad-arduino-argentina-ptec/ [Acceso: 09 – Sep - 2020]

[10] “De Docker a Kubernetes: entendiendo qué son los contenedores y por qué es una de las mayores revoluciones de la industria del desarrollo”, T. Rodríguez, 2019. [Online]. Disponible: https://www.xataka.com/otros/docker-a-kubernetesentendiendo-que-contenedores-que-mayores-revoluciones-industria-desarrollo [Acceso: 10 – Sep - 2020]

[11] “Chronograf 1.8 documentation”. [Online]. Disponible: https://docs.influxdata.com/chronograf/v1.8/ [Acceso: 10 -Sep-2020]

[12] “Get InfluxDB | InfluxData”, InfluxData, 2020. [Online]. Disponible: influxdata.com/get-influxdb/ [Acceso: 10 – Sep - 2020]

[13] “La Plataforma de análisis para todas sus métricas”, Grafana. [Online]. Disponible: https://grafana.com/grafana/ [Acceso: 10 – Sep - 2020]

[14] “¿Conoce qué es un NUC y cómo han evolucionado los computadores hoy en día? | Dynamic Solutions,” Dynamic Solutions, 2016. [Online]. Disponible: dynamicsolutions.com.co/que-es-un-nuc-y-como-han-evolucionado-loscomputadores/#:~:text=NUC%20es%20el%20acr%C3%B3nimo%20de,hasta%20el %20hogar%20o%20se [Acceso: 10-Sep-2020]

[15] “Temperatura de Operación Correcta del Datacenter”, Genesisdata.com. [Online]Disponible:http://picatme.com/pruebas ander/genesis/genesisdata/blog/articulo1.php [Acceso 11-sep-2020]

[16] “Data centers hoy | Protección y administración de datos en la empresa. 2014 [Online]. Disponible: http://www.datacentershoy.com/2014/03/cual-es-la-humedadcorrecta-de-un-data.html [Acceso: 11 – Sep - 2020]

[17] “Docker Introducción”, 2020. [Online]. Disponible: https://aprenderbigdata.com/introduccion-docker/ [Acceso: 14 – Sep - 2020]

[18] “Kit intel NUC NUC7CJYH” [Online]. Disponible: https://www.intel.es/content/www/es/es/products/boards-kits/nuc/kits/nuc7cjyh.html [Acceso: 14 -Sep- 2020]

ANEXOS

ANEXO 1: CÓDIGO ARDUINO MEGA2560/LoRa Shield

```
#include <SPI.h>
#include <RH_RF95.h>
#include <String.h>

RH_RF95 rf95; float frequency = 920.0; // Change the
frequency here.

#define dht_dpin A0 // Use A0 pin to connect the data line of DHT11
byte bGlobalErr; char dht_dat[5];

void setup()
{
  InitDHT();
Serial.begin(9600);
  while (!Serial) ; // Wait for serial port to be available
Serial.println("LoRa_Simple_Client_DHT11");    if
(!rf95.init())
    Serial.println("init failed");
  Serial.println("Humidity and temperature\n\n");
 // Setup ISM frequency
rf95.setFrequency(frequency);    //
Setup Power,dBm
rf95.setTxPower(13);
}
void InitDHT() // Initiate DHT11
{
   pinMode(dht_dpin,OUTPUT);//
   digitalWrite(dht_dpin,HIGH);//Set A0 to output high
}

void ReadDHT() // Read Temperature and Humidity value
{
```

```
   bGlobalErr=0;
byte dht_in;     byte
i;

   pinMode(dht_dpin,OUTPUT);
digitalWrite(dht_dpin,LOW);//pull low data line to send signal.
   delay(30);//Add a delay higher than 18ms so DHT11 can detect the start signal

   digitalWrite(dht_dpin,HIGH);
delayMicroseconds(40);
pinMode(dht_dpin,INPUT);                          //
delayMicroseconds(40);
dht_in=digitalRead(dht_dpin);//Get A0 state
   //   Serial.println(dht_in,DEC);
if(dht_in){        bGlobalErr=1;
return;
      }
   delayMicroseconds(80);//Get DHT11 response , pull low data lineDHT11
dht_in=digitalRead(dht_dpin);
       if(!dht_in){
bGlobalErr=2;
return;
      }
   delayMicroseconds(80);//     for (i=0; i<5; i++)//Ger
Temperature and Humidity value        dht_dat[i] =
read_dht_dat();

   pinMode(dht_dpin,OUTPUT);
digitalWrite(dht_dpin,HIGH);     byte
dht_check_sum =
         dht_dat[0]+dht_dat[1]+dht_dat[2]+dht_dat[3];//Calculate check sum
if(dht_dat[4]!= dht_check_sum)//Error when check sum dismatch
      {bGlobalErr=3;}
};
```

```
byte read_dht_dat(){   byte i = 0;   byte
result=0;   for(i=0; i< 8; i++){
while(digitalRead(dht_dpin)==LOW);//
delayMicroseconds(30);//       if
(digitalRead(dht_dpin)==HIGH)      result
|=(1<<(7-i));//     while
(digitalRead(dht_dpin)==HIGH);//
   }
 return result;
}

void loop()
{
   ReadDHT();     uint8_t data[50] = {0} ;     data[0] = 1 ;     data[1] = 1
;     data[2] = 1 ;// Use Data [0].Data[1], Data[2] to combine a Device
ID.
   data[3] = dht_dat[0]; // store humidity data     data[4] = dht_dat[2];// store
temperature data     rf95.send(data, sizeof(data)); // Send out ID + Sensor
data to LoRa gateway    switch (bGlobalErr)
  {
    case 0:
      Serial.print("Current humdity = ");
      Serial.print(dht_dat[0], DEC);
      Serial.print(".");
      Serial.print(dht_dat[1], DEC);
       Serial.print("%  ");
      Serial.print("temperature = ");
      Serial.print(dht_dat[2], DEC);
      Serial.print(".");
       Serial.print(dht_dat[3], DEC);
     Serial.println("C  ");
     break;
```

```
    case 1:
      Serial.println("Error 1: DHT start condition 1 not met.");
      break;
    case 2:
          Serial.println("Error 2: DHT start condition 2 not
     met.");
     break;
   case 3:
          Serial.println("Error 3: DHT checksum
     error.");       break;
     default:
     Serial.println("Error: Unrecognized code encountered.");
     break;
   }
  delay(4000);
}
```

ANEXO 2: CÓDIGO DRAGINO LG01-P IoT Gateway

```
#include <SPI.h>
#include <RH_RF95.h>
#include <HttpClient.h>

//When use LG01, uncomment this line, so print the result in Console.
//When use LoRa Mini Dev, Comment this link
#define  LG01_GATEWAY

#ifdef LG01_GATEWAY
#include <Console.h>
#include <Process.h>
#define BAUDRATE 115200
#define SerialPrint Console
#else
#define SerialPrint Serial
#endif

float frequency = 920.0;// Change the frequency here.

RH_RF95 rf95; void
setup()
{
  #ifdef LG01_GATEWAY
    Bridge.begin(BAUDRATE);
    SerialPrint.begin();
  #else
    SerialPrint.begin(9600);
  #endif
     if
(!rf95.init())
    {
      SerialPrint.println("LoRa module init failed, Please cehck hardware connection");
while(1) ;
    }
```

```
  //    Setup    ISM    frequency
rf95.setFrequency(frequency);
//          Setup          Power,dBm
rf95.setTxPower(13);
  // Defaults BW Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on
  SerialPrint.print("Listening on frequency: ");
  SerialPrint.println(frequency);
}

void loop()
{
  HttpClient client;


  if (rf95.available())
  {
    // Should be a message for us now        uint8_t
buf[RH_RF95_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(buf);
if (rf95.recv(buf, &len))
    {
     if(buf[0] == 1 & buf[1] == 1 & buf[2] ==1) //Get sensor data from node id 111
      {
        int newData[4] = {0, 0, 0, 0};
for (int i = 0; i < 2; i++)
        {
          newData[i] = buf[i + 3];
        }
        int a = newData[0];         int b
= newData[1];
SerialPrint.print("ID = 111 :");
        SerialPrint.print("Current humdity = ");
        SerialPrint.print(a);
        SerialPrint.print("%  ");
        SerialPrint.print("temperature = ");
        SerialPrint.print(b);
        SerialPrint.println("C  ");
        Console.println("Call Linux Command to Send Data");
```

```
 client.get("http://10.130.1.10:4000/sensores?id=1&temp=" + (String) b + "&hum="
(String) a);
      }
    }
else
    {
        SerialPrint.println("recv failed");
    }
  }
}
```

ANEXO 3: CONFIGURACIÓN DEL DOCKER

```
version: "3.3"

services:
influxdb:
    image: influxdb:1.8
ports:                      -
'8086:8086'     restart:
always     volumes:
- influxdb-storage:/var/lib/influxdb     environment:
- INFLUXDB_DB=sensores
- INFLUXDB_ADMIN_USER=admin
- INFLUXDB_ADMIN_PASSWORD=admin
- INFLUXDB_USER=user
- INFLUXDB_USER_PASSWORD=user

 server:
build:
     context: ./server
ports:                  -
'4000:4000'
restart:        always
depends_on:
- influxdb

 chronograf:
    image: chronograf:1.8
ports:      - 8888:8888
restart: always
environment:
- INFLUXDB_URL=http://influxdb:8086
- INFLUXDB_USERNAME=admin
```

- INFLUXDB_PASSWORD=admin volumes:
- chronograf-storage:/var/lib/chronograf depends_on:
- influxdb

grafana:
image: grafana/grafana:7.0.0
ports: - 3000:3000
restart: always volumes:
- grafana-storage:/var/lib/grafana depends_on:
- influxdb

volumes: grafana storage:
influxdb-storage:
chronograf-storage:

ANEXO 4: CREACIÓN DEL SERVICIO DOCKER CON UN DOCKFILE DEL SERVIDOR REST API

```
FROM node:10

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
COPY package*.json ./

RUN npm install

COPY . .

# Exponer los puertos adecuados
# EXPOSE 8086

# Ejecutars
CMD [ "node", "src/app.js" ]
```

ANEXO 5: CONFIGURACIÓN DEL SERVIDOR

```
const express = require('express') const
sensorRouter = require('./routes/sensores') let
app = express()

app.use(express.json()) // for parsing application/json app.use(express.urlencoded({ extended:
true })) // for parsing application/x-www-form-urlencoded

app.use('/sensores', sensorRouter)

app.use((error, req, res, next) => {
console.log(error)   res.status(500).send('')
})

app.use((req, res) => {
console.log('sin ruta')
res.status(404).send('')
})

let server = app.listen(4000, () => {
console.log('Server is listening on port 4000')
})
```

ANEXO 6: RUTAS PARA INGRESAR SENSORES

```
const express = require('express')

const sensoresController = require('../controller/sensores')

const router = express.Router()

// GET /sensores/ router.get('/',
sensoresController.getSensores)

module.exports = router
```

ANEXO 7: FUNCIÓN CONTROLADORA PARA ENVIAR DATA AL INFLUXDB

```
const { InfluxDB, Point } = require('@influxdata/influxdb-client')

const db = require('../modules/db')

const username = 'admin' const
password = 'admin'

const database = 'sensores' const
retentionPolicy = 'tres_dias'

const bucket = `${database}/${retentionPolicy}`

const clientOptions = {   url:
'http://influxdb:8086',   // url:
'http://localhost:8086',   token:
`${username}:${password}`,
}

const influxDB = new InfluxDB(clientOptions)

const escribirInflux = api => {   return new
Promise((resolve, reject) => {     api
.close()
     .then(() => resolve())
     .catch(error => reject(error))
  })
}

exports.getSensores = async (req, res, next) => {
try {
   let id = req.query.id     let
temperatura = req.query.temp
```

```
 let humedad = req.query.hum


    const dataSensor = db.find(sensor => sensor.id == id)


    let punto = new Point('sensor')
.tag('id', dataSensor.id)
      .tag('codigo', dataSensor.codigo)
      .tag('rack', dataSensor.rack)
      .tag('fila', dataSensor.fila)
      .floatField('temperatura', temperatura)
      .floatField('humedad', humedad)



    let writeAPI = influxDB.getWriteApi('', bucket)


    writeAPI.writePoint(punto)
await escribirInflux(writeAPI)


    res.status(200).send('ok')
} catch(error) {
next(error)
  }
}
```