Desarrollo de un sistema de monitoreo remoto de la calidad del aire en espacios interiores

Cristian Mosquera Escobar

Trabajo de grado para optar por el título de Ingeniero Electrónico

Asesor
Juan Pablo Urrea Duque, PhD

Universidad de Antioquia
Facultad de Ingeniería
Ingeniería Electrónica
Medellín
2023

Cita	(Mosquera Escobar, 2023)
Referencia **Estilo APA 7 (2020)**	Mosquera Escobar, C. (2023). *Desarrollo de un sistema de monitoreo remoto de la calidad del aire en espacios interiores* [Trabajo de grado]. Universidad de Antioquia, Medellín.

UNIVERSIDAD DE ANTIOQUIA
Vicerrectoría de Docencia

Sistema de Bibliotecas

Centro de Documentación Ingeniería (CENDOI)

Repositorio Institucional: https://bibliotecadigital.udea.edu.co

Universidad de Antioquia - www.udea.edu.co

Rector: Jhon Jairo Arboleda Céspedes.

Decano/Director: Jesús Francisco Vargas Bonilla.

Jefe departamento: Augusto Enrique Salazar Jiménez.

Tabla de contenido

Lista de figuras

Siglas, acrónimos y abreviaturas

IAQ	Indoor Air Quality
VOCs	Volatile Organic Compounds
TVOCs	Total Volatile Organic Compounds
PM	Particulate Matter
PM2.5	Particulate Matter 2.5 micrometers
PM10	Particulate Matter 10 micrometers.
IoT	Internet Of Things
I2C	Inter-Integrated Circuit
UART	Universal Asynchronous Receiver-Transmitter
CRC	Cyclic Redundancy Check
AWS	Amazon Web Services
MQTT	MQ Telemetry Transport
QoS	Quality of Service
UTC	Coordinated Universal Time
KMS	Key Management System
EC2	Elastic Compute Cloud
ECS	Elastic Container Service
SQL	Structured Query Language
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
MVC	Model-View-Controller
REST	Representational State Transfer

Resumen

La contaminación del aire es la contaminación del ambiente interior o exterior por cualquier agente químico, físico o biológico que altera la composición de la atmósfera. Aunque existe la percepción de que la mayor parte de la contaminación del aire viene del exterior (automóviles, industria, etc.), la realidad es que el aire en los hogares y oficinas en donde trabajamos puede estar aún más contaminado que el aire exterior. Estos contaminantes pueden provenir de fuentes como productos de limpieza, alfombras, pesticidas, humidificadores, aires acondicionados, moho, etc. Por lo tanto, los riesgos en la salud para muchas personas pueden ser mayores debido a la exposición a la contaminación del aire en este tipo de espacios.

A pesar de que en los últimos años ha habido esfuerzos en Colombia para medir la contaminación del aire, esta se ha centrado mayormente en el monitoreo de la calidad del aire en espacios exteriores. Debido a esto, se hizo necesario desarrollar una estación de monitoreo remoto de la calidad en espacios interiores (IAQ) para establecer las bases de la construcción de una red de monitoreo para este tipo de espacios, en donde se pudieron visualizar los valores de variables como material particulado (PM2.5 y PM10), compuestos orgánicos volátiles (VOCs), CO2, temperatura y humedad. Los datos se recolectaron desde una tarjeta Raspberry Pi y se enviaron por internet al bróker de AWS IoT Core usando el protocolo MQTT y se pudieron visualizar remotamente desde una página web tanto en tiempo real como históricamente en un periodo dado para poder identificar tendencias en el deterioro de la calidad del aire en el espacio monitoreado.

Palabras clave: IAQ, calidad del aire, CO2, VOCs, material particulado, IoT, MQTT.

Abstract

Air pollution is the contamination of the indoor or outdoor environment by any chemical, physical or biological agent that alters the composition of the atmosphere. Although there is a perception that most of the air pollution comes from the outside (vehicles, industry, etc.), the true is that the indoor air in our houses and work offices could be even more polluted than the outdoor air. These pollutants might come from sources like cleaning products, carpets, pesticides, humidifiers, air conditioners, mold, etc. Therefore, health risks can be greater due to the indoor air pollution.

Even though in recent years there have been great efforts in Colombia to measure the air pollution, this focused has been mainly for outdoor air quality. Due to this, it became necessary to develop a station to monitor the indoor air quality (IAQ) remotely and to establish the bases for the construction of a monitoring network for this kind of spaces to measure variables such as particulate matter (PM2.5 and PM10), volatile organic compounds (VOCs), CO2, temperature and humidity. The data was collected from a Raspberry Pi board and sent over the internet to AWS IoT Core using the MQTT protocol, which can be viewed remotely from a browser both in real time and historically over a period to identify trends in the deterioration of air quality in the monitored space.

Keywords: IAQ, Air quality, CO2, VOCs, Particulate matter, IoT, MQTT.

Introducción

El crecimiento del sector industrial ha desencadenado un aumento importante en la emisión de contaminantes, ocasionando que la calidad del aire en las principales ciudades del país se degrade a niveles que afectan la salud de sus habitantes. En Colombia, la mala calidad de aire es el tercer factor generador de costos sociales y para el año 2015 fue la causante de 8.052 muertes en el país, con costos de aproximadamente 12,2 billones de pesos (Ministerio de Ambiente y Desarrollo Sostenible, n.d.). Debido a esto, el gobierno nacional ha venido haciendo esfuerzos para intentar solucionar este problema que incluyen: el desarrollo de políticas para reducir las concentraciones de contaminantes en el aire como el CONPES 3943, el Subsistema de Información Sobre Calidad del Aire –SISAIRE, el Sistema de Alerta Temprana de Medellín y el Valle de Aburrá –SIATA, entre otros.

Dentro de la degradación del aire hay dos componentes principales: la contaminación del aire urbano y la contaminación del aire interior. Esta última generó costos por mortalidad prematura y atención de enfermedades que superaron los $3 billones de pesos en 2015 (Departamento Nacional de Planeación, 2017). Estos contaminantes en espacios interiores pueden provenir de fuentes comunes que tenemos en nuestros hogares y espacios de trabajo como productos de limpieza, pesticidas, humidificadores, aires acondicionados en mal estado o fuentes externas. Ya que los habitantes de las ciudades pasan la mayor parte del tiempo en estos tipos de espacios interiores, se hace necesario tener un sistema de monitoreo que muestre los datos en tiempo real e histórico. Esto con el fin de encontrar tendencias en el deterioro de la calidad del aire para así poder efectuar una investigación y ejecución de acciones preventivas y correctivas como mantenimiento al aire acondicionado, cambio de filtro en purificadores, mejoramiento del sistema de ventilación, etcétera.

La calidad de aire en espacios interiores ha venido tomando gran importancia en países como Estados Unidos y Alemania los cuales han empezado a implementar normas y guías para el monitoreo en este tipo de espacios. Debido a esto, han surgido compañías que se especializan en diseñar y comercializar productos de monitoreo de la calidad del aire en espacios interiores. A pesar de que en Colombia hay redes de la calidad del aire en exteriores administradas por entidades gubernamentales, no hay redes de monitoreo oficiales para espacios interiores. Por lo tanto, la motivación principal de este trabajo de grado es establecer una base de código abierto

para la construcción de una red de monitoreo en Colombia para este tipo de espacios el cual se propone solamente desde el trabajo de grado y no está ligado aún a ninguna empresa u entidad gubernamental.

En este proyecto se desarrolló una estación de monitoreo remoto de la calidad del aire en espacios interiores usando una tarjeta Raspberry pi la cual recolecta los datos usando sensores de CO2, PM2.5, PM10, compuestos orgánicos volátiles (VOCs), temperatura y humedad. Estos datos se envían por Wi-fi usando el protocolo MQTT al bróker AWS IoT Core los cuales son almacenados en una base de datos en AWS Timestream. Estos datos se pueden visualizar en tiempo real desde un navegador web usando gráficos de línea, al igual que los datos históricos en un intervalo de tiempo ingresado por el usuario. A pesar de que solo se construyó una estación física de monitoreo, el sistema puede escalarse para conectar múltiples estaciones a la misma red en diferentes localizaciones las cuales también pueden consultarse desde el aplicativo web para así construir una red de monitoreo.

1 Objetivos

1.1 Objetivo general

Desarrollar una estación de monitoreo de la calidad del aire en espacios interiores con visualización de datos en tiempo real e históricos desde un navegador web.

1.2 Objetivos específicos

- Recolectar datos de material particulado PM2.5 y PM10 usando el sensor PMS5003 para su posterior visualización y para dar una idea al usuario de los niveles de exposición en el espacio monitoreado de acuerdo con la resolución 2254 de 2017 la cual establece los niveles permisibles de contaminantes en el aire, incluidos PM2.5 y PM10.
- Recolectar datos de CO2 usando el sensor SGP30 para su posterior visualización y para comprobar que exista una adecuada ventilación en el lugar en donde se encuentra la estación.
- Recolectar datos de compuestos orgánicos volátiles (VOCs) usando el sensor SGP30 para su posterior visualización y para dar una idea al usuario de las cantidades de productos químicos nocivos a los cuales están expuestas las personas en el espacio monitoreado.
- Recolectar datos de temperatura y humedad usando el sensor BME280 para su posterior visualización y para tener una idea del adecuado nivel de confort térmico para los usuarios en el espacio monitoreado.
- Enviar los datos de los sensores a través de Wifi usando el protocolo MQTT sobre una conexión segura para almacenarlos y visualizarlos remotamente desde un aplicativo web.
- Almacenar los datos recibidos de la estación en una base de datos en la nube para su posterior consulta en un rango de tiempo determinado por el usuario.
- Visualizar los datos enviados en tiempo real e histórico de cada sensor remotamente desde un aplicativo web usando el framework Angular y la librería Chart.js para

comprender tendencias y patrones en la calidad del aire en el espacio monitoreado y así poder efectuar medidas correctivas.

- Validar el funcionamiento de la estación al someterla a gases contaminantes como humo de estaño, polvo, alcoholes, limpiadores y desinfectantes. Las gráficas en el aplicativo deberán mostrar picos en los datos reportados por la estación durante la exposición a estos gases y químicos.

2 Marco teórico

2.1 Calidad del aire en espacios interiores (IAQ)

IAQ por sus siglas en inglés (Indoor Air Quality) se refiere a la calidad del aire dentro y alrededor de edificios y estructuras, especialmente en lo que se refiere a la salud y la comodidad de los ocupantes de estos tipos de espacios. Algunos efectos a la salud pueden aparecer unos pocos minutos luego de la exposición a diferentes contaminantes y estos incluyen irritación de los ojos, la nariz y la garganta, dolores de cabeza, mareos y fatiga. Pero también pueden generar enfermedades a largo plazo más severas como enfermedades respiratorias, enfermedades del corazón y cáncer.

Las fuentes contaminantes que liberan gases o partículas son la causa principal de los problemas de la calidad del aire interior al igual que una ventilación inadecuada puede aumentar los niveles de contaminantes ya que no se tiene suficiente aire del exterior para diluir las emisiones de las fuentes interiores. Los altos niveles de temperatura y humedad también pueden aumentar las concentraciones de algunos contaminantes (US Environmental Protection Agency, 2021).

Entre las fuentes contaminantes más frecuentes están: el tabaco, asbestos, pisos, tapizados o alfombras recién instalados, productos de limpieza, aires acondicionados en mal estado, humidificadores, etc.

2.2 Compuestos orgánicos volátiles (VOCs) y totales (TVOCs)

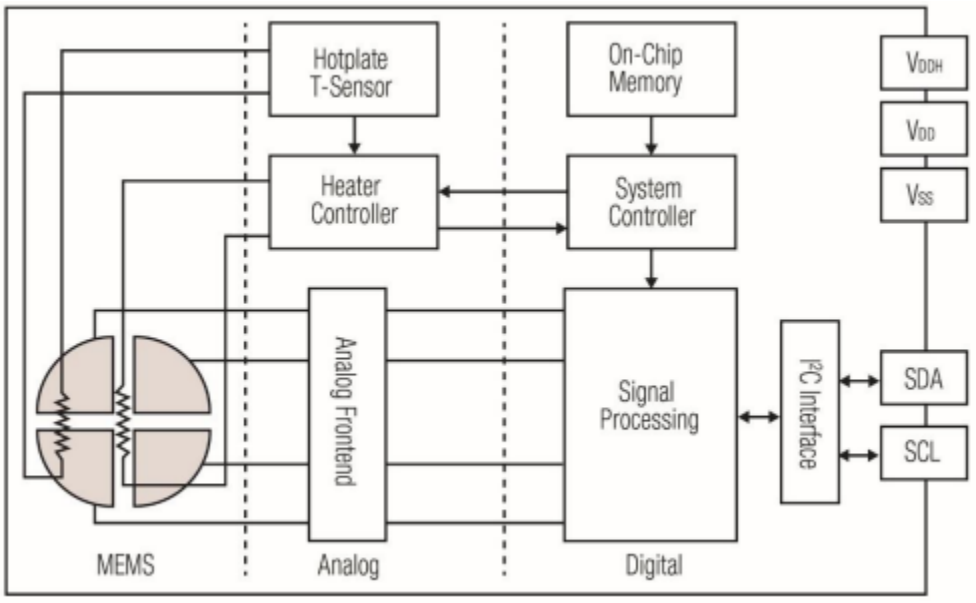
Son gases emitidos por algunos productos los cuales incluyen químicos que pueden ocasionar efectos a la salud a corto y largo plazo. Las concentraciones de VOCs son generalmente mayores en espacios interiores y son emitidos por una gran variedad de productos como pinturas, pesticidas, desinfectantes, humidificadores, aromatizadores, etc. Algunos de los efectos a la salud son: dolor de cabeza, mareos, daño en los riñones, daños al sistema nervioso central, irritación en ojos e incluso cáncer (US Environmental Protection Agency, 2021). Ya que los VOCS son una familia de gases y es muy difícil monitorearlos individualmente, se ha adoptado una medida conocida como TVOCs (Compuestos orgánicos volátiles totales) para

medir la cantidad total de VOCS en un espacio dado. Su medida se da usualmente en partes por billón (ppb).

2.3 Sensor de gas SGP30

El sensor SGP30 es un sensor de gas digital con dos señales de salida para CO2 y TVOCS diseñado para la fácil integración con sistemas purificadores de aire, ventiladores controlados y aplicaciones basadas en IoT. El sensor es de tipo MOS (Metal Oxide gas Sensor) el cual detecta algunos tipos de gases midiendo el cambio de resistencia por absorción de gases con un rango de medida para TVOCS de 0 ppb a 60000 ppb y de 400 ppm a 60000 ppm para CO2. El diagrama del sensor SGP30 (Figura 1) consiste en una interfaz digital I2C, una microplaca controlada por temperatura y dos señales de calidad del aire interior (CO2 y TVOCs) preprocesadas (Sensirion, 2020).

Figura 1
Diagrama funcional del sensor SGP30



Nota. Fuente SGP30 Datasheet (Sensirion, 2020)

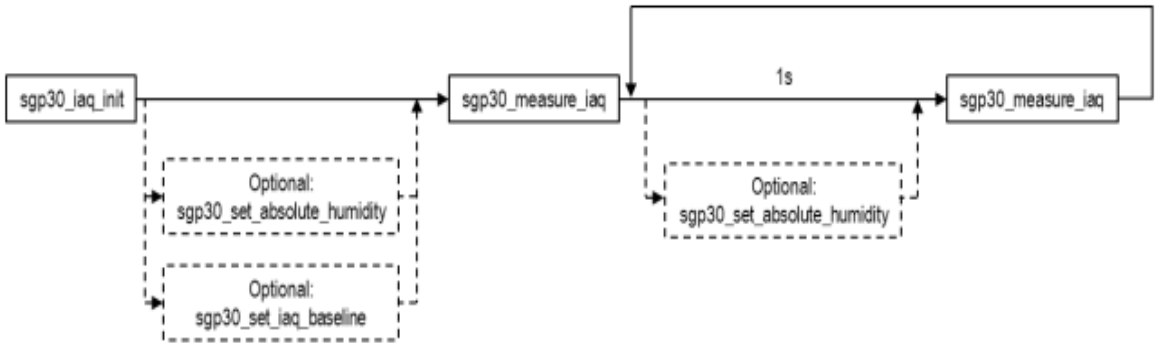
El protocolo de comunicación con el sensor funciona de la siguiente manera: primero se debe enviar una trama tipo START (el cual incluye la dirección del dispositivo) seguido de un

comando de 16 bits llamado sgp30_iaq_init (0x2003) y otro comando de 16 bits llamado sgp30_measure_iaq (0x2008). El sensor responde con 2 bytes de datos y un byte de CRC por cada una de las dos señales de calidad de aire (TVOCS y CO2). Se debe enviar un nuevo comando sgp30_iaq_init después de cada encendido.

Opcionalmente el sensor acepta el comando llamado sgp30_set_absolute_humidity (0x2061) el cual se usa para realizar una compensación de humedad y lograr que las lecturas de TVOCs y CO2 sean mucho más precisas. El diagrama de secuencia de comandos (Figura 2) describe los pasos necesarios para realizar la lectura del sensor usando el protocolo I2C.

Figura 2

Secuencia de comandos I2C para realizar la lectura del sensor SGP30



Nota. Fuente SGP30 Datasheet (Sensirion, 2020)

Para este prototipo se usó la placa preensamblada por Adafruit (Figura 3) basada en el mismo chip SGP30 la cual contiene los conectores y demás componentes soldados haciéndola muy conveniente para diseñar prototipos.

Figura 3
Placa Adafruit SGP30

Nota. Fuente Adafruit SGP30 (Adafruit, n.d.)

2.4 CO2

Es un gas producto de la respiración humana y se mide en partes por millón (ppm). A pesar de ser un gas no tóxico, en concentraciones de 2500 ppm a 5000 ppm puede causar dolor de cabeza y en concentraciones mayores a 100.000 ppm puede causar pérdida de conocimiento (Greiner, 1991). El CO2 es comúnmente usado como indicador de la insuficiencia en los sistemas de ventilación en los edificios. Para obtener las lecturas de CO2 se usó el mismo sensor de gas SGP30 ya que este también entrega las medidas de CO2 por medio de una comunicación I2C.

2.5 Material particulado (PM)

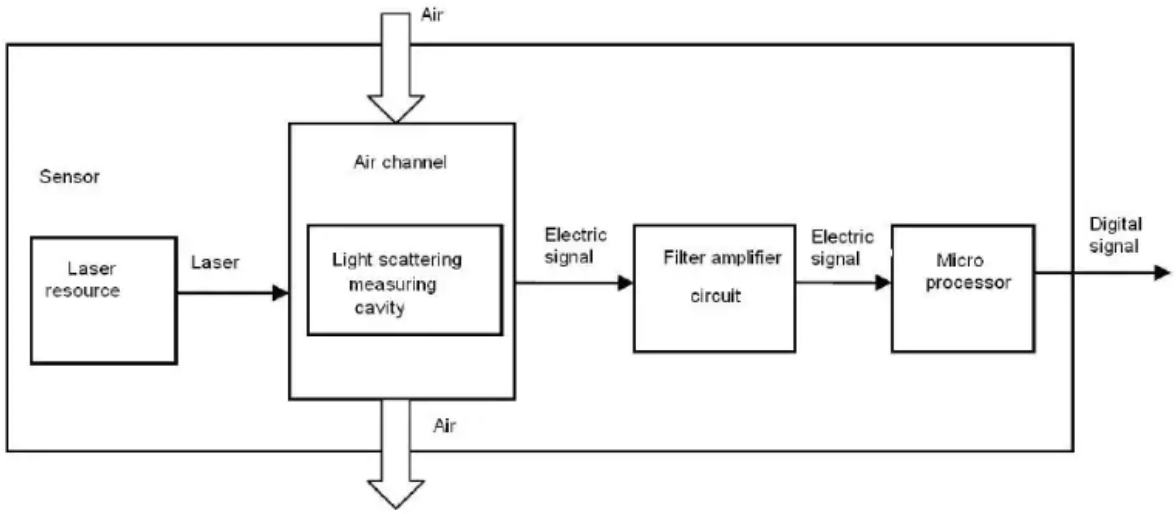
Son partículas sólidas o líquidas de tamaño microscópico que se encuentran en el aire y al ser inhaladas pueden causar serios problemas a la salud. Estas se identifican dependiendo del tamaño, siendo PM10 (menor o igual a 10 micrómetros) y PM2.5 (menor o igual a 2.5 micrómetros) las más monitoreadas debido a los efectos que pueden causar a la salud (US Environmental Protection Agency, 2021). Como referencia, el tamaño de una partícula de 2.5 micrómetros es aproximadamente 30 veces más pequeño que el diámetro de un cabello humano.

Algunos de los efectos a la salud causados por estas partículas incluyen: degradación en la capacidad pulmonar, asma, irritación en las vías respiratorias, tos seca, etcétera. Según la norma colombiana, los niveles máximos permisibles de PM10 y PM2.5 para un tiempo de exposición de 24 horas son 75 g/m3 y 37 g/m3 respectivamente (Ministerio De Ambiente y Desarrollo Sostenible, 2017)

2.6 Sensor de material particulado PMS5003

Se usó el sensor PMS5003 el cual usa el principio de dispersión para obtener el diámetro y el número de partículas por unidad de volumen de material particulado. Este principio consiste en usar un láser el cual apunta a una pequeña cavidad por donde circula el aire. Las partículas suspendidas en dicho aire causan una variación angular de la intensidad de la luz en el rayo que las atraviesa generando una señal eléctrica (Figura 4) la cual es procesada internamente por el módulo PMS5003 generando una señal digital compatible con el protocolo UART (Yong, 2016).

Figura 4
Diagrama funcional del sensor PMS5003



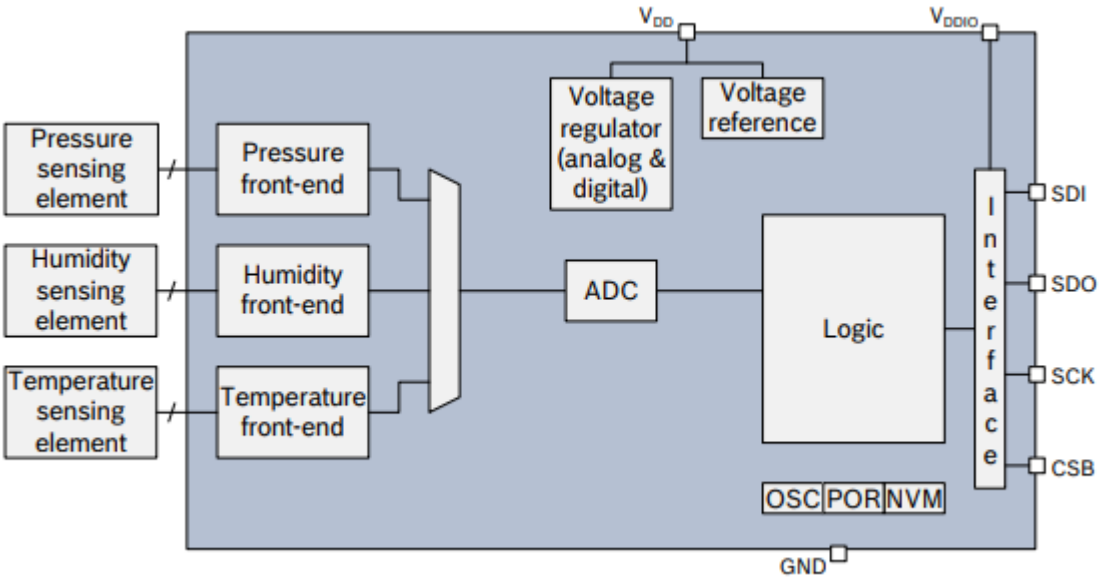
Nota. Fuente PMS5003 series data manual (Young, 2016)

2.7 Sensor de temperatura, presión y humedad BME280

Es un sensor digital de temperatura, presión y humedad (Figura 5) de bajo consumo pensado para aplicaciones como automatización de hogares, celulares, relojes fitness, juegos, etc. El sensor es compatible con el protocolo I2C y tiene un rango de operación de -40 a 80 °C y de 0 a 100% de humedad relativa (BOSCH, 2022).

Figura 5

Diagrama funcional del sensor BME280



Nota. Fuente BME280 datasheet (BOSCH, 2022)

El sensor almacena los valores medidos en unos registros internos los cuales se deben leer usando el protocolo I2C. La figura 6 muestra las tramas necesarias para la lectura de los registros 0xF6 y 0xF7. Los valores de temperatura se deben leer de los registros 0xFA, 0xFB y 0xFC mientras que los de humedad se deben leer de los registros 0xFD y 0xFE.

Figura 6

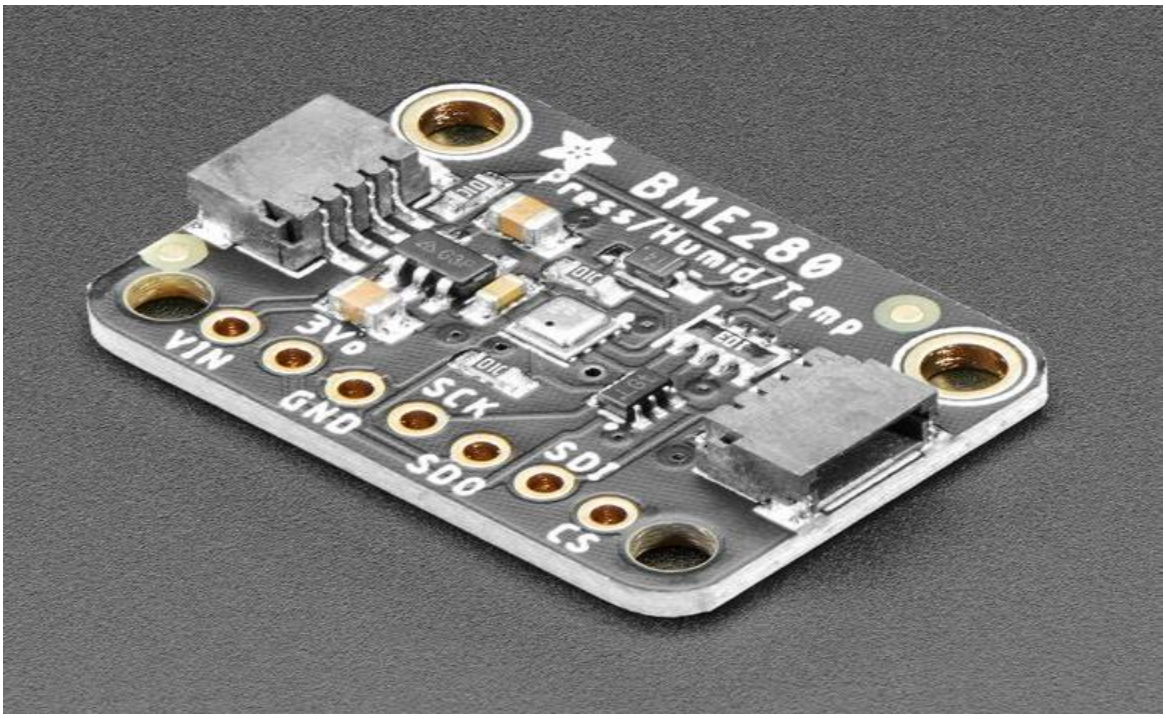
Secuencia de comandos I2C para realizar la lectura del sensor BME280

Nota. Fuente BME280 datasheet (BOSCH, 2022)

Para este prototipo se usó la placa bme280 preensamblada de Adafruit (Figura 7) la cual contiene los conectores y demás componentes soldados lo cual la hace conveniente para el diseño de prototipos. La figura 8 muestra esta placa preensamblada.

Figura 7

Placa Adafruit BME280



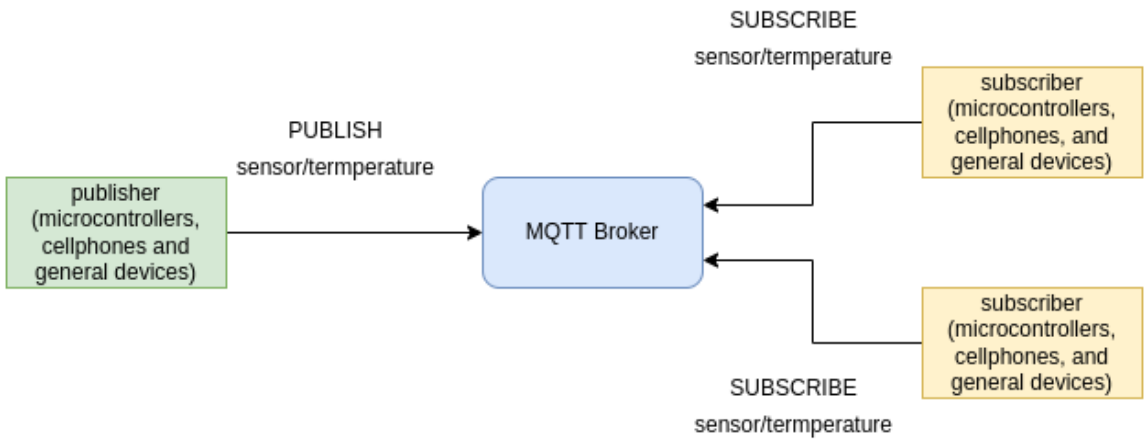
Nota. Fuente Adafruit BME280 (Adafruit, n.d.)

2.8 MQTT y AWS IoT Core

MQTT es un protocolo liviano de mensajería estándar pensado para aplicaciones de tipo IoT (Internet of Things). Debido a que usa poco ancho de banda, su uso es muy común en dispositivos con poco poder de procesamiento como microcontroladores. Este protocolo está basado en un mecanismo de tipo suscriptor/publicador el cual provee un bajo nivel de acople entre dispositivos y está compuesto por uno o varios canales por los cuales los publicadores envían mensajes y dichos mensajes son recibidos por los suscriptores a ese canal específico (OASIS, 2019).

La figura 8 muestra un diagrama simplificado de los componentes implicados en este tipo de comunicación. El publicador publica datos con un canal llamado *topic* al cual otros clientes llamados suscriptores se pueden suscribir para recibir el dato. El *bróker* es el encargado de administrar las suscripciones y de enrutar los mensajes de los publicadores a los suscriptores.

Figura 8
Componentes que hacen parte de la especificación MQTT



El protocolo soporta varios niveles de calidad de servicio (QoS). El nivel QoS 0 significa que el mensaje se entrega de acuerdo con las capacidades de la red (el receptor no envía ninguna respuesta y el publicador no realiza ningún reintento). Esto significa que el mensaje llega al receptor una vez o nunca. Es el tipo de comunicación más rápida, pero debe usarse en redes robustas y en aplicaciones que permitan pérdida de datos. El nivel QoS 1 garantiza que el

mensaje se entregue al menos 1 vez, lo cual significa que el mensaje llega al receptor una o más veces. Es un poco más lenta ya que involucra más tramas en la comunicación y sólo debe usarse en aplicaciones que permiten datos repetidos. El nivel QoS 2 garantiza que el mensaje se entregue exactamente 1 vez. Esta es la más alta calidad de servicio y se debe usar cuando no es aceptable la pérdida ni la duplicación de mensajes. Es el nivel más lento e implica un aumento considerable en los gastos generales asociados con esta calidad de servicio (tiempo de procesamiento, memoria y ancho de banda).

Para este proyecto se usó el *bróker* de AWS llamado AWS IoT Core, el cual soporta la especificación MQTT 3.1.1. Debido a que este es un servicio cloud, se minimiza la cantidad de infraestructura a configurar y administrar. AWS provee SDKs para usar desde diferentes dispositivos y soporta diferentes lenguajes de programación como C, C++, Python, Java y JavaScript. Entre sus características está la comunicación usando certificados para que la comunicación entre dispositivos sea segura, y la integración con otros servicios de AWS para el enrutamiento de mensajes para su almacenamiento en alguna base de datos (Amazon Web Services, n.d.).

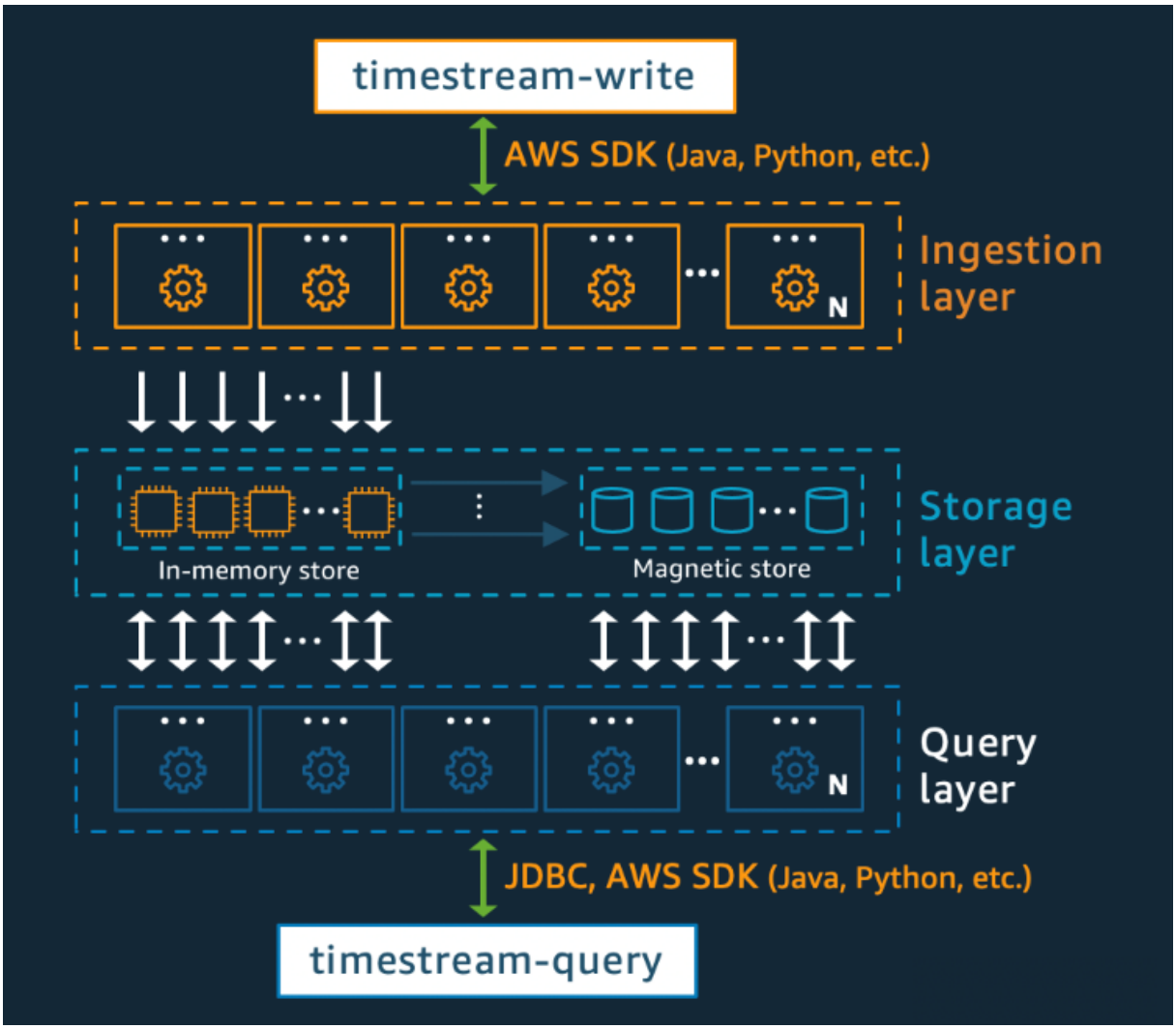
2.9 AWS Timestream

Es una base de datos optimizada para datos de series de tiempo lo cual la hace perfecta para aplicaciones de tipo IoT. Es un 90% más barata que las bases de datos relacionales convencionales y puede llegar a ser 1000 veces más rápida (Amazon Web Services, n.d.). Consiste en un atributo llamado *Dimensión* el cual describe los metadatos de una serie temporal. Generalmente se usan atributos que identifican de forma única los datos de una serie temporal como por ejemplo el id del dispositivo asociado a los datos. La medida (*measurement*) es el valor que está siendo medido, como el valor de la temperatura y el atributo llamado *timestamp* el cual indica cuando se recopiló la medida.

AWS Timestream contiene dos tipos de almacenamiento: In-memory store (RAM) y Magnetic Store. In-memory store es un almacenamiento temporal donde se guardan los datos más recientes para que las consultas sean más rápidas para luego ser enviados al almacenamiento magnético el cual es más lento, pero mucho más barato para el almacenamiento de datos por

periodos largos de tiempo (meses o años). La figura 9 muestra un diagrama simplificado de la arquitectura de AWS Timestream.

Figura 9
Arquitectura de AWS Timestream



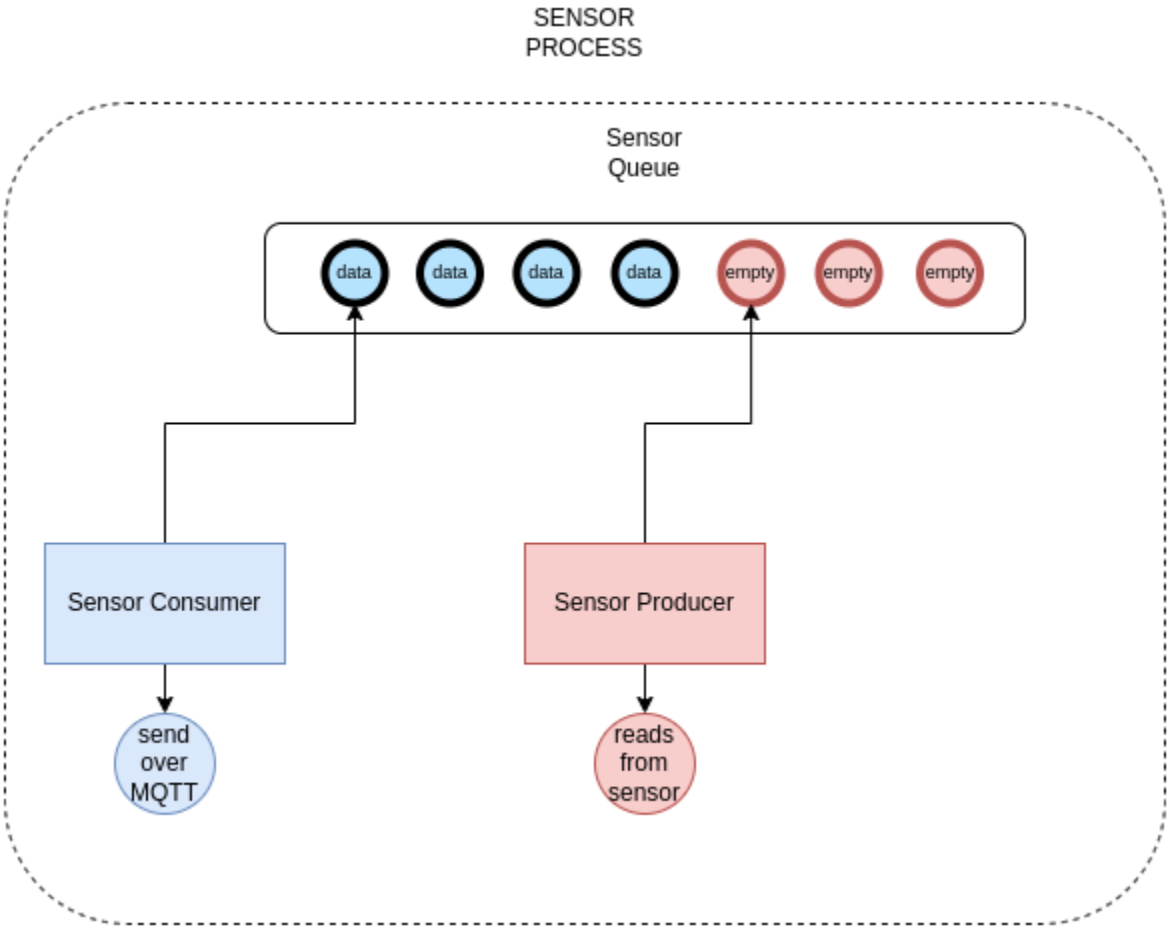
Nota. Fuente What is Amazon Timestream? (Amazon Web Services, n.d.)

3 Metodología

3.1 Lectura de datos de los sensores desde la Raspberry Pi

Se usó el patrón de diseño llamado consumidor y productor el cual consiste en uno o más hilos llamados productores los cuales generan datos que son almacenados en una cola de mensajes. Estos datos son luego consumidos por uno o más hilos llamados consumidores para ser procesados. La figura 10 muestra los componentes que hacen parte de este patrón.

Figura 10
Patrón de diseño Productor-Consumidor

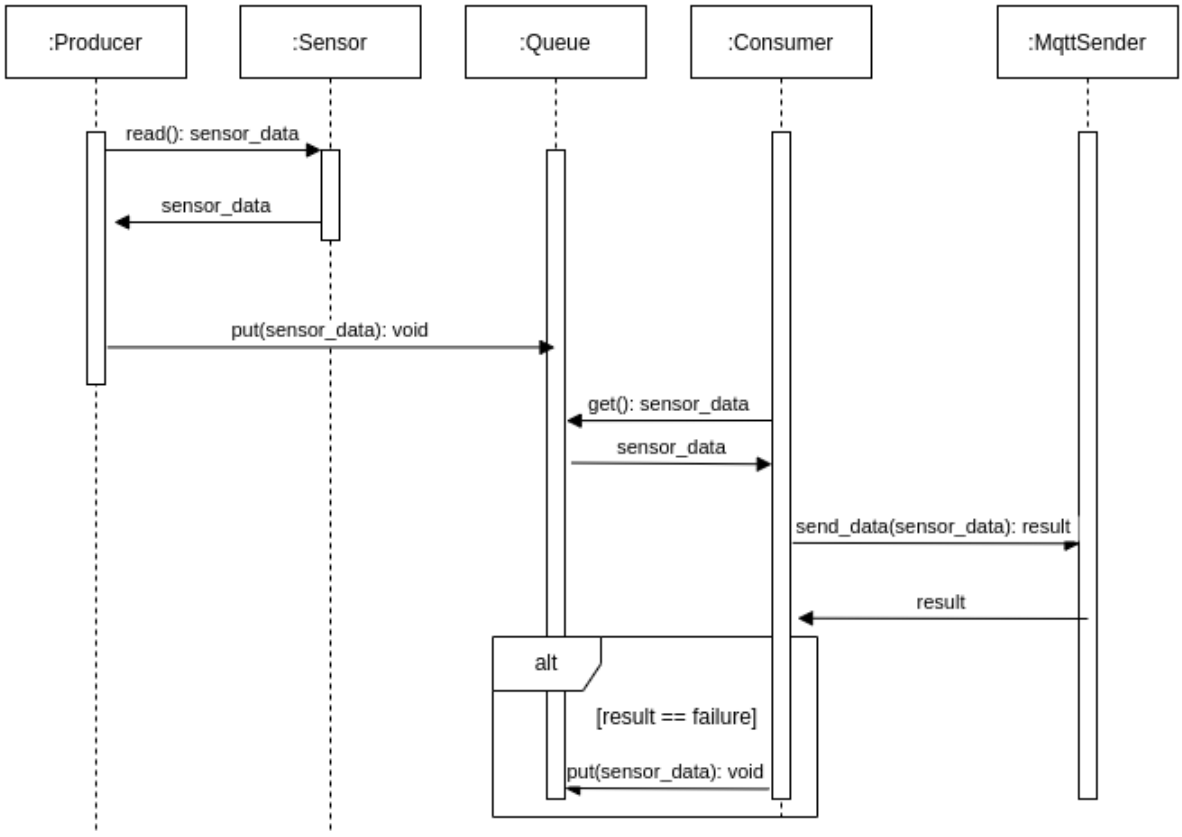


Cada sensor tiene dedicados su propio productor, consumidor y una cola de datos para poder lograr el envío en paralelo de cada sensor. El productor realiza la lectura del sensor cada 5 segundos.

La figura 12 muestra el diagrama de secuencia el cual describe el procedimiento de la lectura y envío de los datos del sensor. El productor lee el dato del sensor y lo almacena en una cola. Luego, este dato es luego leído por el consumidor el cual lo intenta enviar por medio de MQTT. Si por alguna razón el envío falla, el dato es puesto de nuevo en la cola para ser enviado luego. Esto garantiza que no haya pérdida de datos durante una interrupción en la conexión a internet.

Figura 11

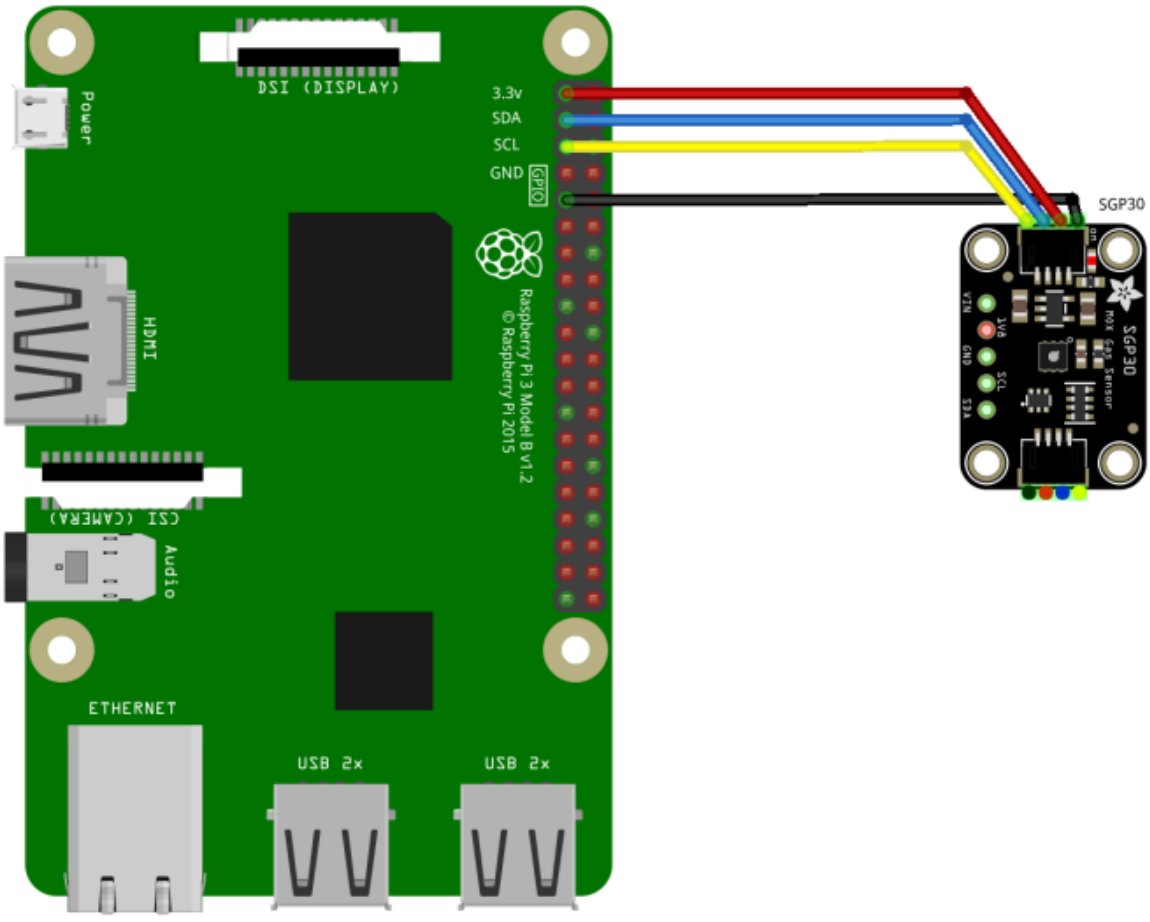
Diagrama de secuencia Productor-Consumidor



3.1.1 Comunicación entre la Raspberry pi y el sensor SGP30

Para la establecer la comunicación I2C entre la Raspberry y el sensor se usó la librería llamada Adafruit-Blinka la cual contiene varios paquetes que contienen las interfaces para interactuar con los pines de propósito general (I/O) de la Raspberry, así como las implementaciones de los diferentes protocolos como SPI, I2C, UART, etc. La figura 12 muestra el diagrama de conexiones entre la Raspberry y la tarjeta SGP30 para establecer la comunicación por medio de I2C. El sensor SGP30 usa la dirección 0x58 la cual es usada por la Raspberry pi para establecer la comunicación con este sensor.

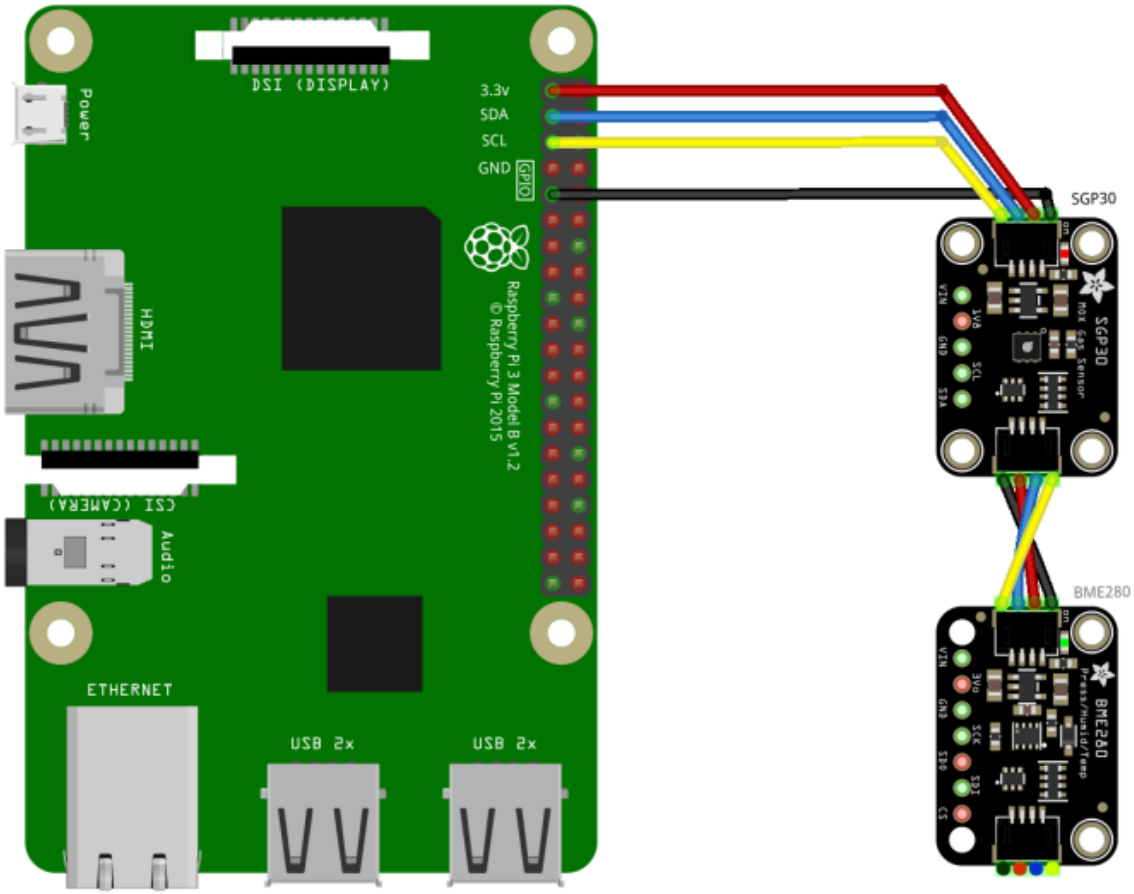
Figura 12
Diagrama de conexión entre la Raspberry Pi y el sensor SGP30



3.1.2 Comunicación entre la Raspberry pi y el sensor BME280

Nuevamente se requirió el uso de la librería Adafruit-Blinka para establecer la comunicación I2C entre la Raspberry y el sensor ya que el sensor BME280 también soporta el protocolo digital I2C. En este caso, el sensor BME280 usa la dirección 0x77 lo cual nos permite conectarlo al mismo bus de datos que el sensor SGP30. La Raspberry actúa como elemento primario que decide de cual sensor quiere leer los datos usando la dirección del sensor. La figura 13 muestra el diagrama de conexiones entre la Raspberry y los sensores SGP30 y BME280.

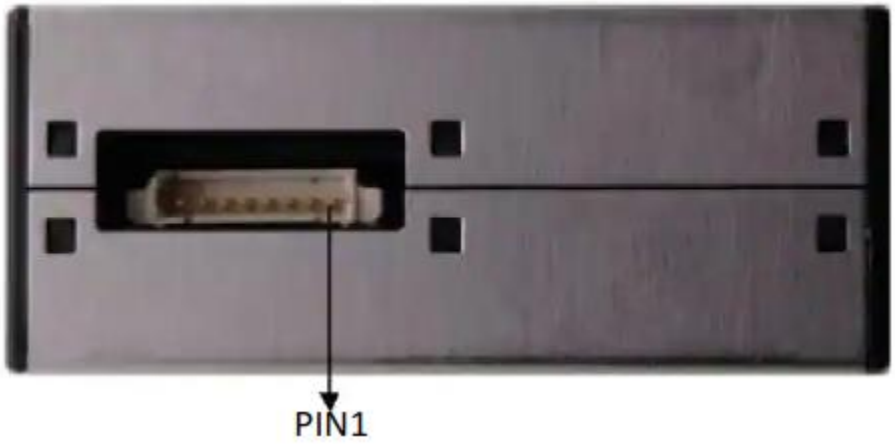
Figura 13
Diagrama de conexión entre la Raspberry Pi y los sensores SGP30 y BME280



3.1.3 Comunicación entre la Raspberry pi y el sensor PMS5003

Nuevamente se requirió el uso de la librería Adafruit-Blinka para establecer la comunicación UART entre la Raspberry y el sensor PMS5003. La figura 14 muestra el tipo de conector usado por el sensor PMS5003 en donde el pin 1 es el conector positivo (+5V), el pin 2 es el conector negativo (GND), el pin 4 es el conector del receptor (RX) y el pin 5 es el pin del transmisor (TX).

Figura 14
Conector del sensor PMS5003

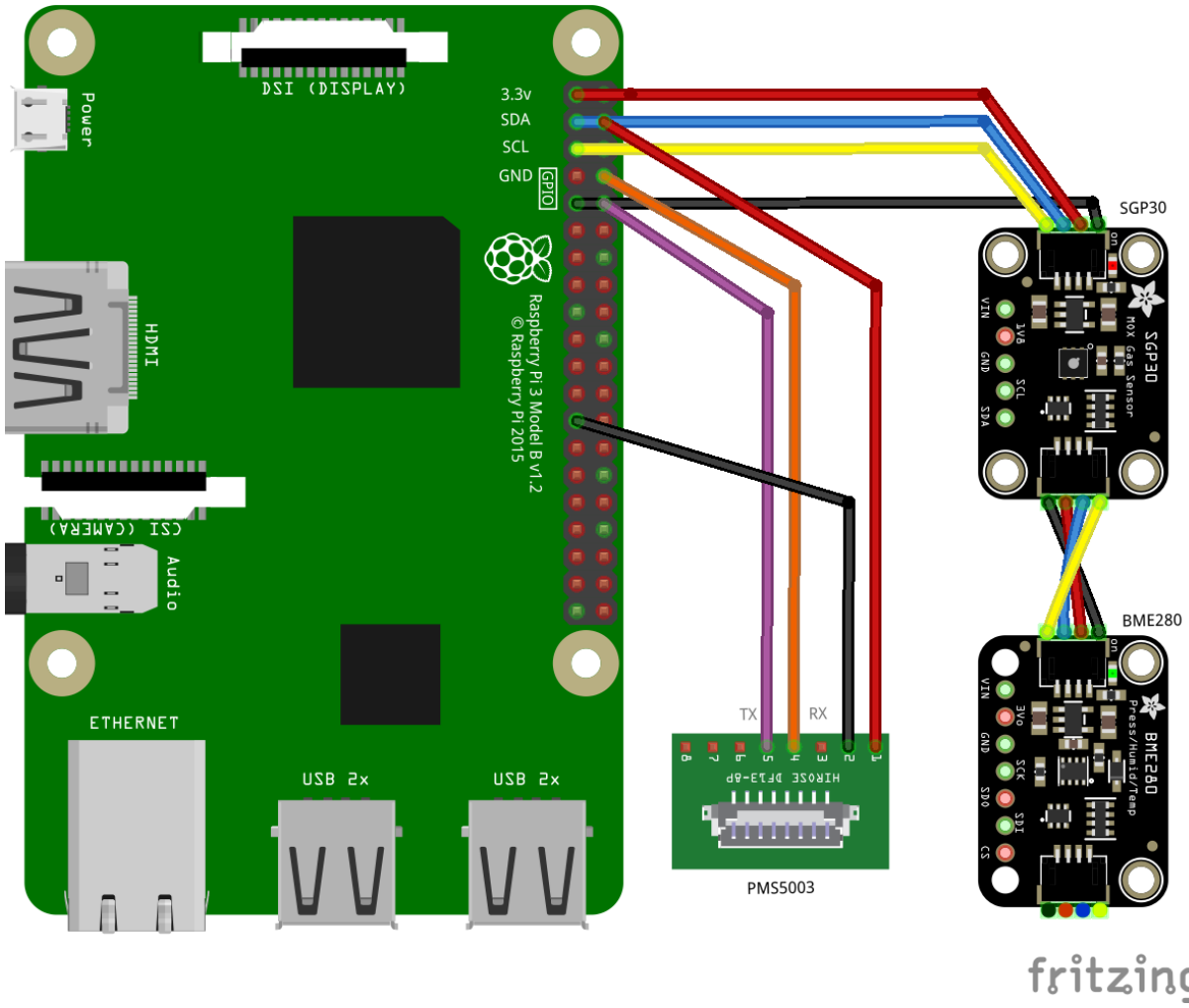


Nota. Fuente PMS5003 series data manual (Young, 2016)

El diagrama con la conexión entre la Raspberry pi y los sensores SGP30, BME280 y PMS5003 se muestra en la figura 15.

Figura 15

Diagrama de conexión entre la Raspberry Pi y los sensores SGP30, BME280 y PMS5003



3.2 Configuración en AWS IoT Core

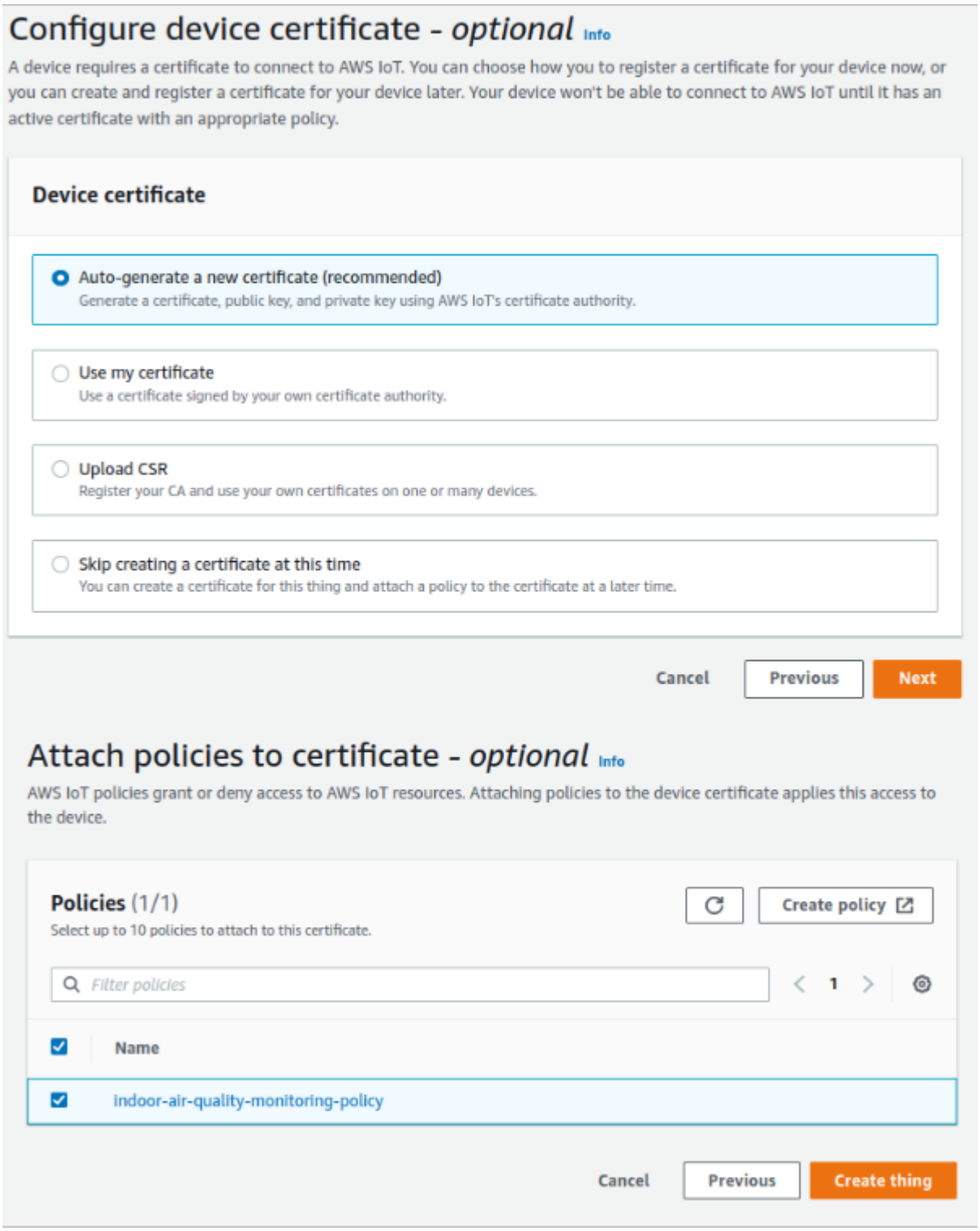
Debido a que los recursos creados en AWS son protegidos por defecto, se hizo necesario crear varias políticas de permisos para permitir las conexiones a ciertos tópicos en el bróker de AWS. Una política (policy, en inglés) en AWS es un objeto que cuando es asociado a un recurso, define sus permisos. Estos permisos determinan cuando cierta acción es permitida o no sobre el recurso. El siguiente código define la estructura en tipo JSON que se desarrolló para permitir conexiones a cada uno de los tópicos. Aunque aquí solo se muestra la estructura que se desarrolló para el tópico llamado temperature, se usó la misma política para cada uno de los tópicos.

```
{
        "Version": "2012-10-17",
        "Statement": [
            {
             "Effect": "Allow",
             "Action": [
              "iot:Publish", "iot:Receive", "iot:Subscribe", "iot:Connect"
              ],
            "Resource": [
              "arn:aws:iot:us-east-1:<account-id>:topic/sensor/temperature",
            ]
          }
        ]
}
```

Una vez creada la política, se procedió a realizar el registro del dispositivo en AWS IoT Core. La representación de un dispositivo específico en AWS IoT Core se llama Thing. El registro del dispositivo se hizo desde la consola de AWS en la sección AWS IoT > Manage > Things > Create Thing > Create Single Thing. La figura 16 muestra los datos usados para realizar el registro.

Figura 16

Formulario de registro de dispositivo en AWS IoT Core



Una vez terminado el registro de nuestro thing, AWS generó un kit de conexión que consiste en varios certificados de tipo X.509 los cuales se usaron desde la Raspberry para autenticar el dispositivo y lograr la conexión a AWS IoT Core. Estos certificados se pueden renovar o revocar en caso de que sean comprometidos.

3.3 Conexión y envío de mensajes a AWS IoT Core desde la Raspberry Pi

Luego de haber creado nuestro Thing en AWS, se usó el SDK de Amazon el cual contiene un conjunto de librerías que facilitó la conexión al bróker. Se usó el método mqtt_connection_builder el cual nos permite crear conexiones usando la URL y los certificados anteriormente proveídos por AWS. El siguiente código muestra un ejemplo de cómo se usó el builder para lograr esta conexión.

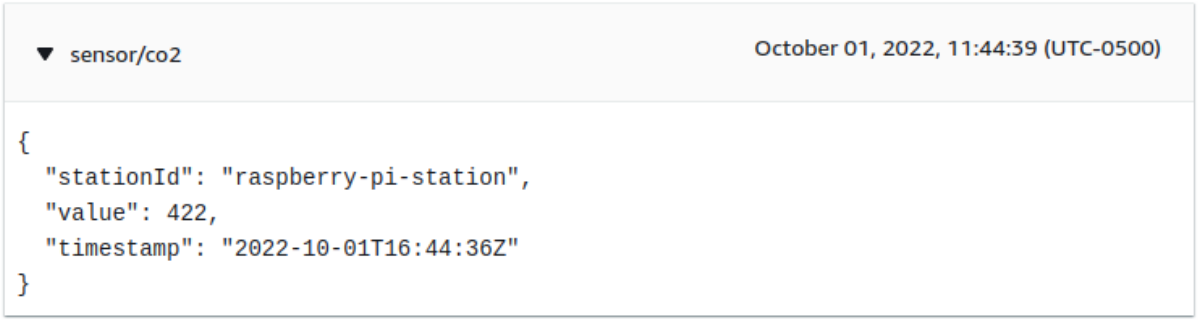
```
mqtt_connection = mqtt_connection_builder.mtls_from_path(
            endpoint=mqtt_config.host,
            port=mqtt_config.port,
            cert_filepath=mqtt_config.cert_filepath,
            pri_key_filepath=mqtt_config.private_key_filepath,
            ca_filepath=mqtt_config.ca_filepath,
            client_id=mqtt_config.station_id,
            clean_session=False,
            keep_alive_secs=mqtt_config.keepalive,
            http_proxy_options=None)
```

Una vez lograda la conexión a AWS IoT core desde la raspberry pi, se procedió a realizar el envío del mensaje usando nuestro objeto mqtt_connection. El siguiente código muestra un ejemplo de cómo se realiza el envío del mensaje.

```
mqtt_connection.publish(topic, payload, qos=mqtt.QoS.AT_LEAST_ONCE)
```

En este caso, el método publish acepta un topic, un payload y un QoS. El topic determina a cuál tópico se quiere enviar el mensaje (temperatura, humedad, CO2, etc), el payload es el mensaje que se quiere enviar y el QoS (Quality of Service) determina la garantía de entrega de un mensaje específico. En este caso, se usó el valor 1 para el QoS el cual especifica que el mensaje sea recibido al menos 1 vez. Esto garantiza que no se pierdan los mensajes en tránsito, aunque este tipo de comunicación es más lenta debido a que involucra más tramas en el protocolo MQTT (tramas de tipo ACKNOWLEDGED). Para comprobar el envío del mensaje, se usó el MQTT test client en la consola de AWS el cual nos permite ver los mensajes que se envían a un tópico específico. La figura 17 muestra cómo llegan los mensajes al tópico CO2.

Figura 17
Ejemplo de datos recibidos por el bróker AWS IoT Core

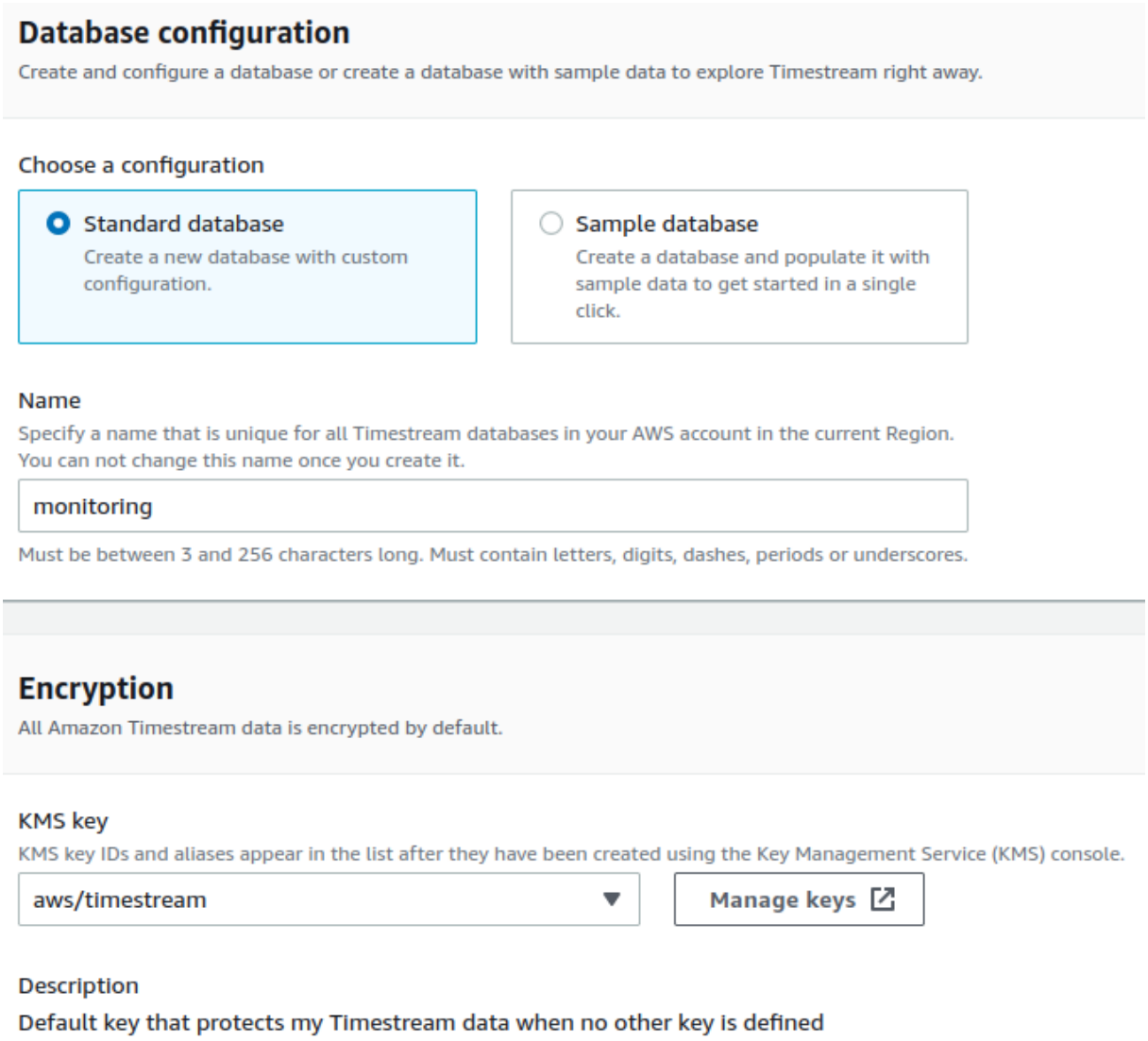


La estructura JSON usada en el mensaje consiste en 3 atributos. El atributo *stationId* contiene el nombre o id de la estación que reportó la lectura del sensor, el atributo *value* contiene el valor medido del sensor (CO2 en este caso) y el atributo *timestamp* contiene la hora en la cual se realizó la lectura usando la zona horario universal (UTC).

3.4 Creación de la base de datos y tablas en AWS Timestream

Para el almacenamiento de los datos, se creó una base de datos y una tabla por cada sensor en AWS Timestream, desde la consola de AWS en la sección Timestream > Databases > Create database. La figura 18 muestra la interfaz para la creación de la base de datos.

Figura 18
Formulario para la creación de la base de datos

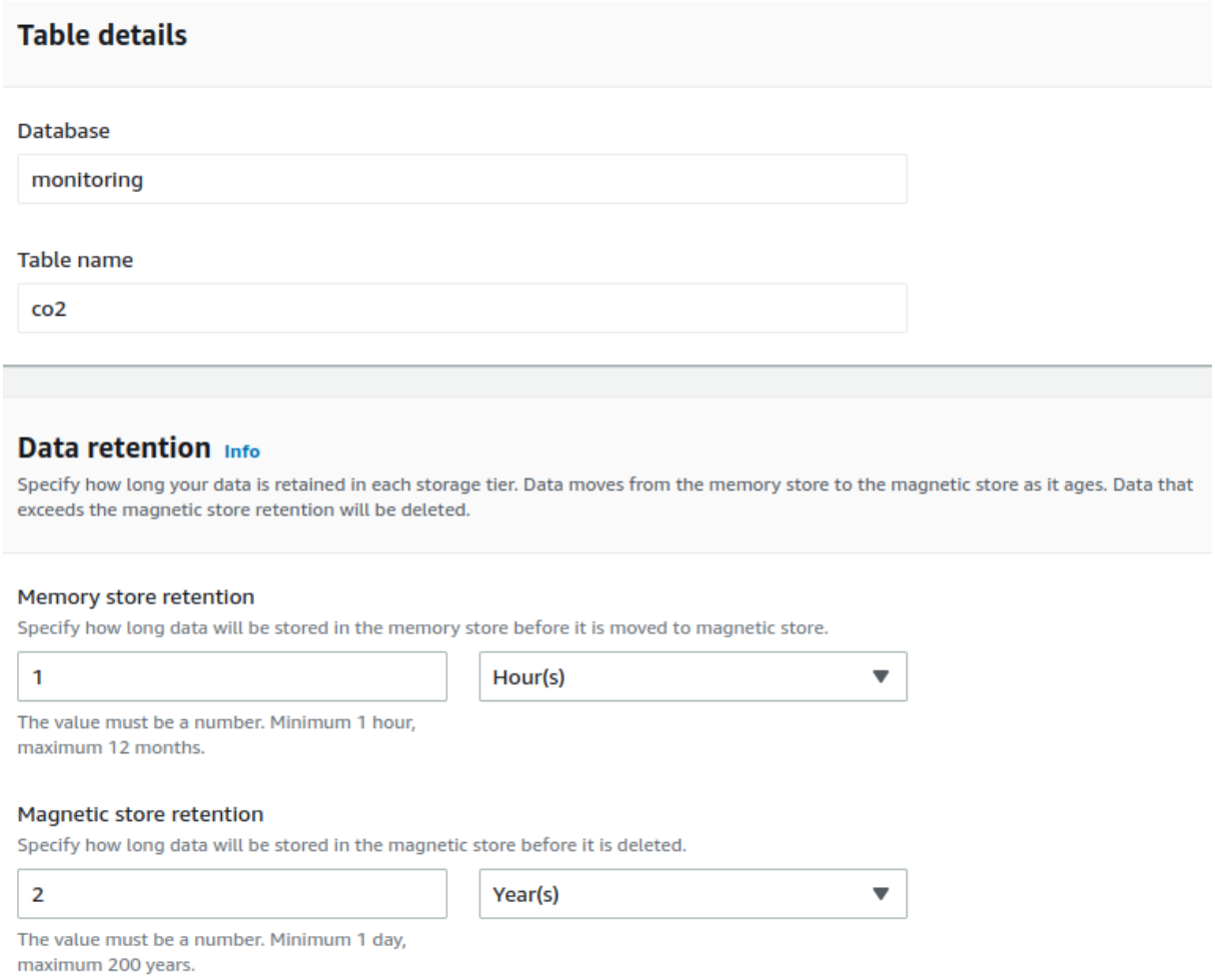


En la interfaz definimos el nombre que en nuestro caso es *monitoring* y una llave (KMS key) la cual permite la encriptación de datos. La encriptación de datos provee una protección extra en caso de que nuestra base de datos sea comprometida, ya que es necesario usar la llave para desencriptar dichos datos allí almacenados.

Luego de haber creado la base de datos, se procedió a crear las tablas para cada sensor (temperatura, humedad, VOC, CO2, PM10 y PM2.5). La figura 19 muestra la interfaz para la creación de la tabla de CO2.

Figura 19

Formulario para la creación de la tabla de CO2 en Timestream



Allí se seleccionó la base de datos anteriormente creada llamada monitoring, el nombre de la tabla (CO2) y el tiempo de retención. Para la retención en memoria RAM se usó el valor de 1 hora, esto significa que los datos almacenados en la última hora son almacenados en este espacio y su consulta es mucho más rápida. Luego de 1 hora estos datos son enviados al almacenamiento magnético el cual es más lento de consultar, pero es más barato y tiene una retención de 2 años en nuestro caso.

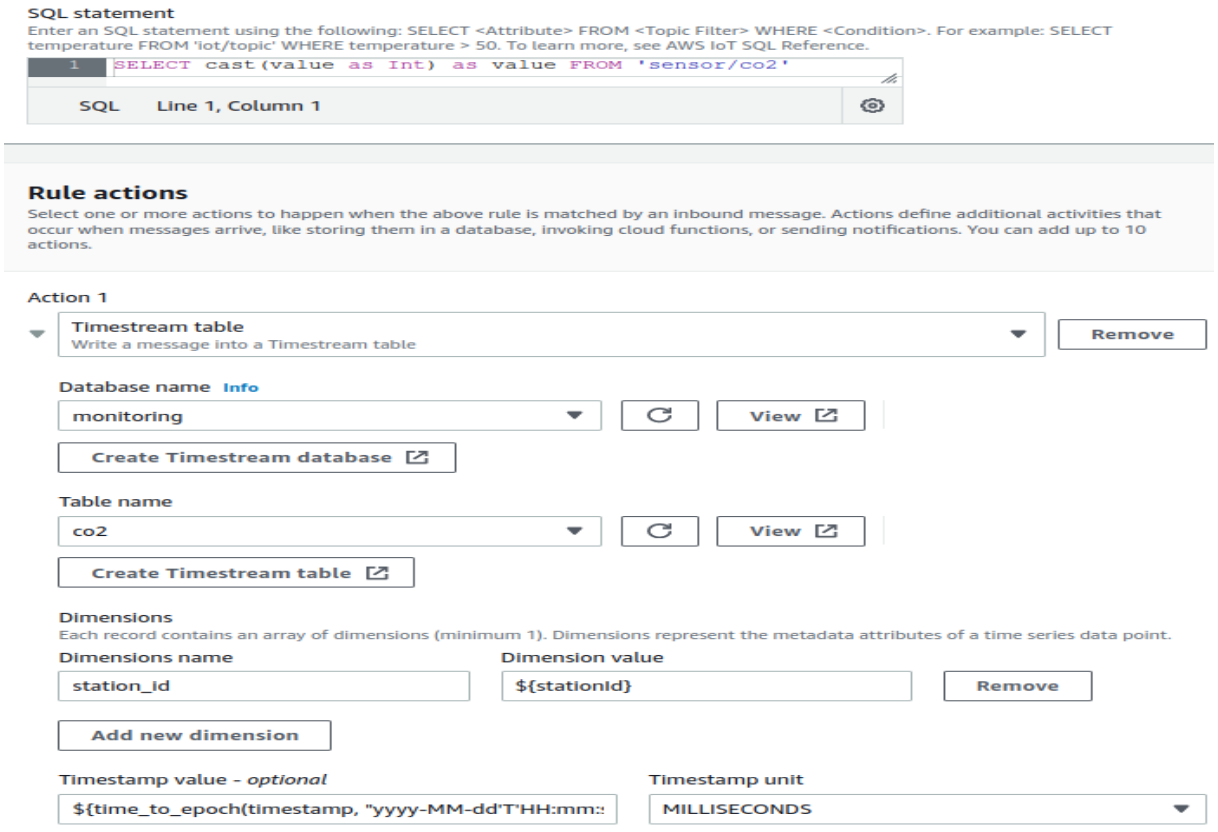
3.5 Almacenamiento de los datos en AWS Timestream

Para almacenar los datos recibidos por el bróker de AWS IoT Core en la base de datos, se usó AWS IoT Rules las cuales definen un conjunto de acciones que se ejecutan cuando los datos son recibidos por el bróker de AWS IoT Core. Estas reglas soportan acciones como: almacenar los datos en una base de datos, generar alarmas y alertas, enviar los datos a un servidor web, etc.

En nuestro caso, se crearon unas reglas para enviar los datos a nuestra base de datos en AWS Timestream previamente creada. Para crear estas reglas, se usó la consola de AWS en la sección AWS IoT > Message Routing > Rules. La figura 20 muestra la interfaz para la creación de la regla para almacenar los datos de CO2 en la base de datos.

Figura 20

Formulario para la creación de las reglas de almacenamiento a AWS Timestream desde IoT Core



El primer paŕametro es una sentencia SQL para extraer la lectura del sensor del JSON en el mensaje MQTT. Luego se define la acción, que en nuestro caso es el envío a AWS Timestream usando la base de datos monitoring y la tabla co2. Como dimensión, se usa el atributo stationId del JSON en el mensaje MQTT y en el timestamp se usa la función time_to_epoch para convertir el timestamp en formato ISO del JSON a milisegundos. El proceso se repitió para cada sensor usando la tabla adecuada para almacenar los datos.

Para validar el correcto funcionamiento de estas reglas, se debió consultar la base de datos para comprobar que los datos de los sensores recibidos por el bróker fueron almacenados en AWS Timestream. La figura 21 muestra el resultado del query para la tabla co2. Allí se confirmó la correcta configuración entre AWS IoT Core y Timestream para el almacenamiento de los datos.

Figura 21

Datos en AWS Timestream para la tabla de CO2

station_id	time	measure_value::bigint
raspberry-pi-station	2022-10-02 02:42:47.000000000	1184
raspberry-pi-station	2022-10-02 02:42:52.000000000	1169
raspberry-pi-station	2022-10-02 02:42:57.000000000	1123
raspberry-pi-station	2022-10-02 02:43:02.000000000	1120
raspberry-pi-station	2022-10-02 02:43:07.000000000	1148
raspberry-pi-station	2022-10-02 02:43:12.000000000	1139
raspberry-pi-station	2022-10-02 02:43:17.000000000	1198

3.6 Diseño de la API tipo REST para la consulta de datos históricos

Para la consulta de los datos almacenados en la base de datos se expuso una API tipo REST. Esta API provee una interfaz para realizar consultas en un periodo de tiempo dado para un sensor en específico. La API se escribió usando el lenguaje de programación JAVA junto con el framework Spring MVC el cual nos facilitó el desarrollo de la API.

La API se invoca usando una petición HTTP tipo GET y tiene la siguiente forma:

/station/${stationId}/metric/${sensorId}/historic?start=${startDate}&end=${endDate}.

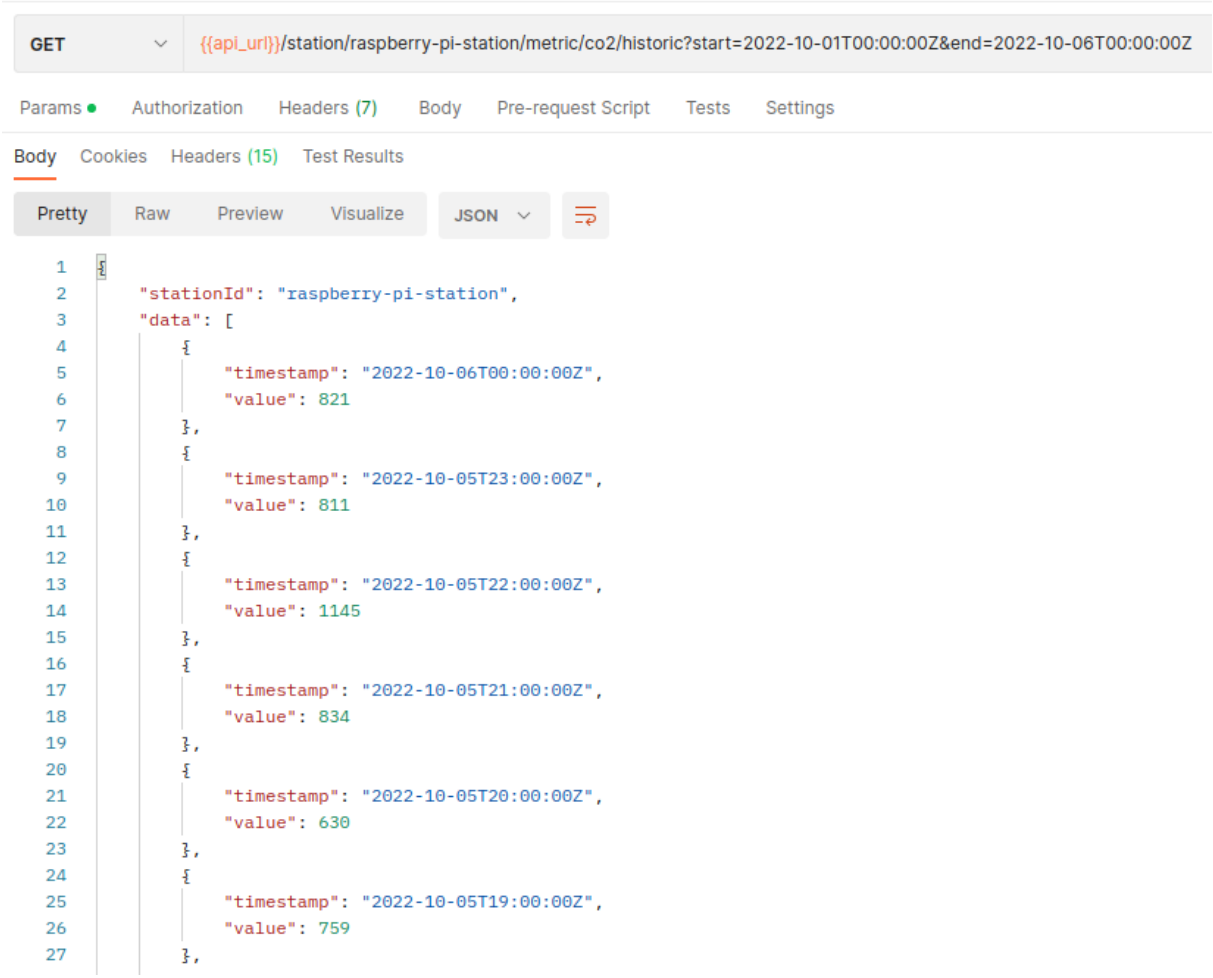
El parámetro ${stationId} hace referencia al id de la estación que se quiere consultar. En nuestro caso, solo tenemos una estación que envía datos, aunque el sistema es escalable y se pueden conectar más estaciones con diferente id. El parámetro ${sensorId} hace referencia al sensor que se quiere consultar. Los valores permitidos son temperature, humidity, vocs, co2, pm25 y pm10. Los parámetros ${startDate} y ${endDate} son el periodo de tiempo en el cual se quieren consultar los datos y deben ser de la forma YYYY-MM-DD:'HH:mm:ss y en tiempo UTC. El resultado retornado por la API es un JSON con la siguiente forma:

```
{
  stationId: string,
  data: Array [
    {
      timestamp: Date,
      value: number
    }
  ]
}
```

La figura 22 muestra el resultado de la petición HTTP la cual retorna los datos de CO2 de la estación con id raspberry-pi-station entre el 1 de octubre de 2022 a las 12 am y el 6 de octubre de 2022 a las 12 am.

Figura 22

Ejemplo de petición HTTP GET para la consulta de datos históricos de CO2



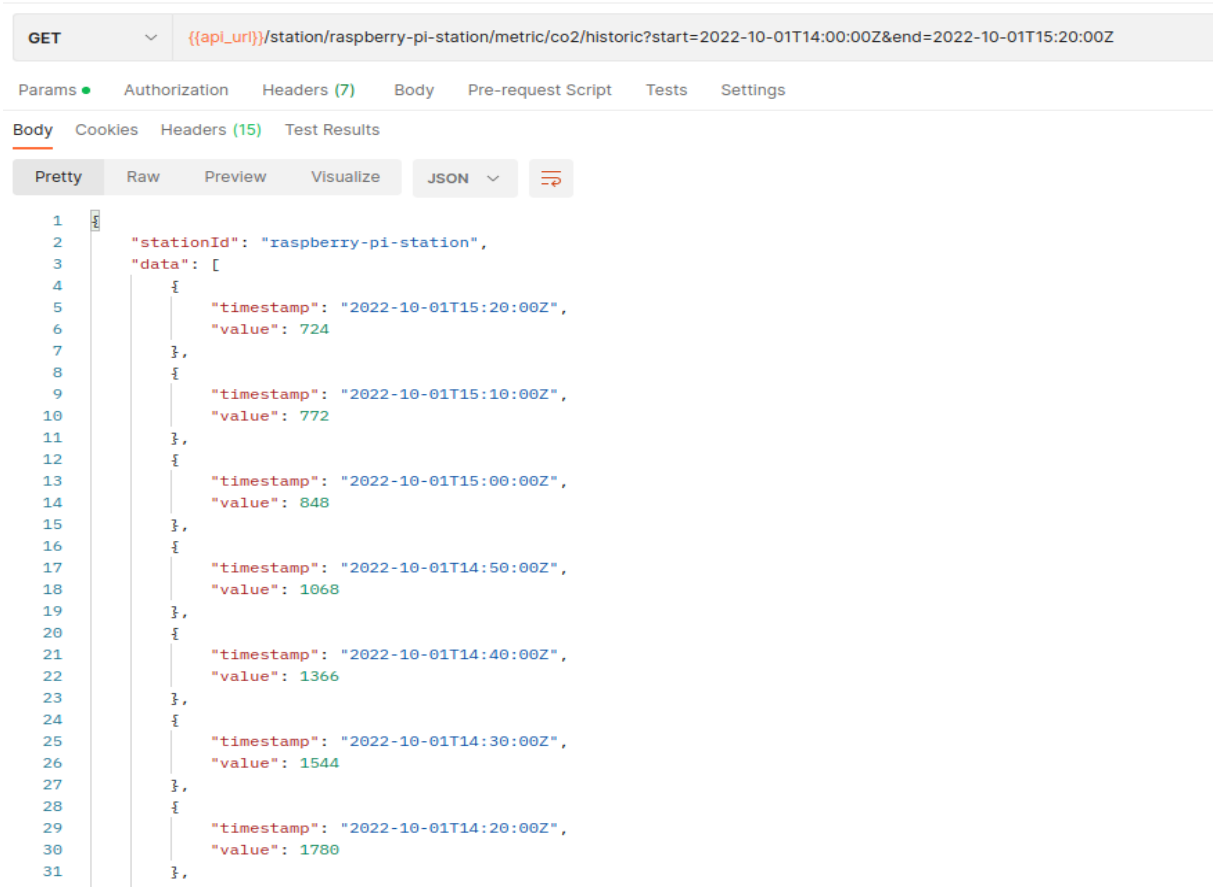
Debido a que la Raspberry envía datos cada 5 segundos por sensor (12 datos por minuto por cada sensor), las consultas de datos históricos para un lapso de tiempo muy grande tomaban demasiado tiempo para ser procesadas debido a la gran cantidad de datos que se debe retornar desde la API. Por ejemplo, si se quiere consultar en un lapso de 30 días para la métrica CO2, la cantidad de datos a retornar por la API usando la granularidad de 5 segundos es: 12 datos x 60 min x 24 h x 30 días = 518400 datos por sensor lo cual es demasiado para un lapso de tan solo 30 días. Esto hizo que el procesamiento de los datos tomara aproximadamente 10 segundos para este caso particular, ocasionando una mala experiencia de usuario al usar la API o el cliente web que grafica los datos. Para solucionar este problema se debió incrementar la granularidad en proporción al intervalo de tiempo de la siguiente forma:

- Para los intervalos menores a 1 hora, se usó la granularidad original de 5 segundos. Esto garantiza la máxima precisión en los datos ya que la API retorna todos los datos generados en este lapso para un total de 720 datos para un solo sensor.
- Para los intervalos entre 1 hora y 1 día, se usó la granularidad de 10 minutos. Es decir, que se calcula el valor promedio por cada intervalo de 10 minutos. Por ejemplo, para un lapso de 1 día se retorna un total de 144 datos (6 datos / hora = 144).
- Para los intervalos 1 día y 30 días, se usó la granularidad de 1 hora. Nuevamente se calcula el valor promedio por cada intervalo de 1 hora. Por ejemplo, para un lapso de 30 días se retorna un total de 720 datos.
- Para los intervalos mayores a 1 mes, se usó la granularidad de 1 día. Por ejemplo, para un lapso de 1 año se retorna un total de 365 datos.

Manteniendo la cantidad de datos por debajo de 1000, se evidenció una mejora importante en el tiempo de procesamiento en la API y el cliente WEB. La figura 23 muestra un ejemplo de una petición para un lapso de 1hora y 20 minutos. Allí se puede apreciar como la API retorna el promedio de CO2 en periodos de 10 minutos.

Figura 23

Ejemplo de petición HTTP GET para la consulta de datos históricos de CO2 en un lapso de 1 hora y 20 minutos



Para obtener los datos almacenados en AWS Timestream se usó el SDK de Amazon para el lenguaje JAVA. La forma de consultar datos en AWS Timestream se hizo usando sentencias SQL. El SDK contiene los métodos que nos permitió conectarnos a Timestream y así poder consultar los datos en un rango de tiempo determinado. El siguiente código muestra un ejemplo de cómo podemos consultar datos en AWS Timestream usando el cliente TimestreamClient y un query en forma de String.

```
String Query = "....";
QueryRequest queryRequest = QueryRequest.builder().queryString(query).build();
QueryIterable queryResponse = timestreamClient.queryPaginator(queryRequest);
```

La figura 24 muestra un ejemplo del resultado de la sentencia SQL que genera el código en JAVA para realizar una consulta a AWS Timestream en un intervalo de tiempo. Allí se puede ver cómo se agrupan los resultados en lapsos de tiempo usando la función BIN y calculando el promedio con la función AVG.

Figura 24

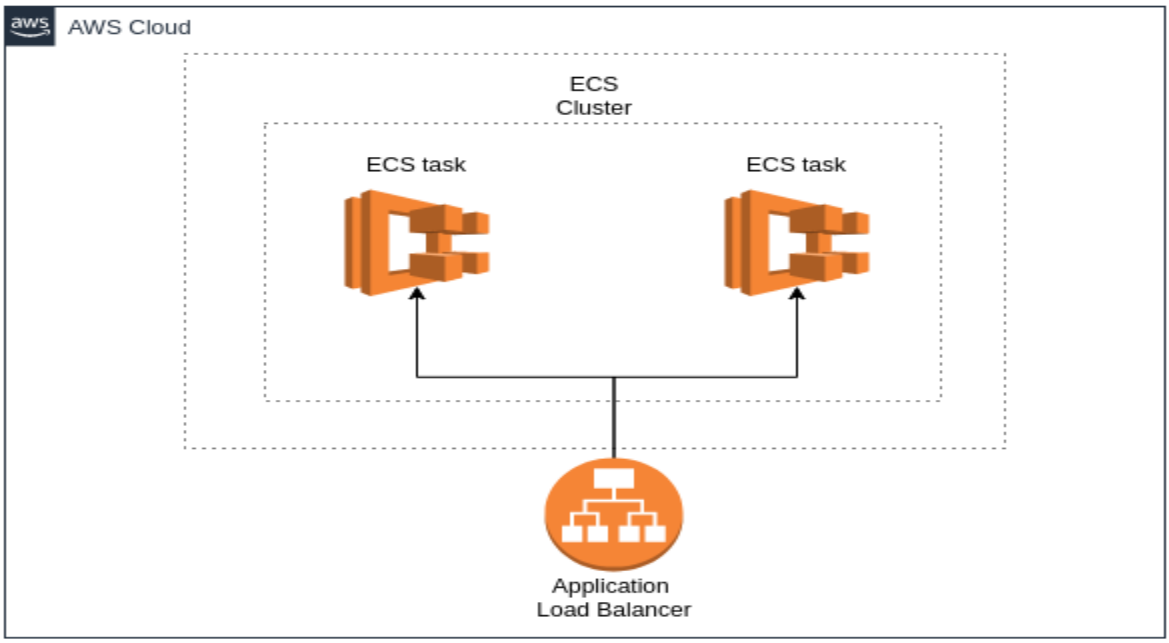
Ejemplo de sentencia SQL para realizar la consulta en un intervalo de tiempo

```
SELECT station_id, to_milliseconds(BIN(time, 600s)) AS timestamp,
cast(ROUND(AVG(measure_value::bigint), 2) as Int) AS value
FROM "monitoring"."co2"
    WHERE station_id = 'raspberry-pi-station'
    AND (time between from_milliseconds(1664632800000) and from_milliseconds(1664637600000))
GROUP BY station_id, BIN(time, 600s) ORDER BY timestamp DESC
```

3.7 Despliegue de la API en AWS usando ECS

El despliegue de esta API se hizo usando AWS Elastic Container Service y un balanceador de carga la cual se describe en la figura 25. El código de la API se ejecuta en un contenedor que se conoce como ECS Task el cual se puede escalar a múltiples instancias dependiendo de qué tan alto sea el tráfico que recibe la aplicación desde el exterior. Cada instancia es manejada por un balanceador de carga el cual recibe el tráfico desde el exterior y reparte las peticiones de forma uniforme entre cada una de las instancias de la aplicación que se ejecutan en ECS.

Figura 25
Diagrama de componentes de AWS que conforman la API



3.7.1 Creación del balanceador de carga

La creación de este balanceador de carga se hizo desde la interfaz de AWS en la sección EC2 > Load Balancers > Create Application Load Balancer. La figura 26 muestra algunos de los atributos usados en la creación del balanceador

Figura 26
Atributos usados para la creación del balanceador de carga

Name	iaq-alb
ARN	arn:aws:elasticloadbalancing:us-east-1:354562611480:loadbalancer/app/iaq-alb/60b1519f6c1c3d2d
DNS name	iaq-alb-1908252564.us-east-1.elb.amazonaws.com (A Record)
State	Active
Type	application
Scheme	internet-facing
IP address type	ipv4
	Edit IP address type
VPC	vpc-0dc11f9ed2d390a22

Los atributos importantes por destacar son la VPC (Virtual Private Cloud) la cual nos provee un entorno privado virtual. Por defecto AWS nos provee una VPC cuando se crea una cuenta y el valor del DNS el cual se usó para invocar la API.

3.7.2 Creación del Servicio en ECS

Primero se generó una imagen usando Docker con el código fuente de JAVA. Está imagen de Docker es un contenedor que empaqueta el código y todas las dependencias necesarias para ejecutar la aplicación en diferentes entornos. El código en el contenedor siempre se ejecutará de la misma manera, independientemente de la infraestructura, es decir, el contenedor garantiza que el código funcione de manera uniforme a pesar de las diferencias entre sistemas operativos (Windows, Linux, o macOS). Luego procedimos a crear la infraestructura en ECS.

Luego se creó un Task Definition en la sección ECS > Task Definitions > Create. Este task definition sirve como esqueleto para crear copias (tasks) de la aplicación que queremos ejecutar. La figura 27 muestra algunos de los atributos usados en la creación del task definition. Algunos de los atributos importantes a mencionar son la imagen de Docker que se quiere usar y los valores de CPU y RAM.

Figura 27
Atributos usados para la creación del task definition

	Container Name …	Image	CPU Units	GPU…	Inference A…	Hard/Soft memory limits (MiB)
▾	iaq-container…	354562611480.dkr.ecr.us-east…	256			512/--

Luego se creó el servicio en ECS en la sección ECS > Clusters > Service> Create. Las figuras 28 y 29 muestran algunos de los atributos usados en la creación del servicio. Algunos de los atributos importantes a mencionar son el task definition, la referencia al balanceador de carga y el número de instancias de nuestro servicio que se van a ejecutar.

Figura 28
Atributos usados para la creación del servicio en ECS - parte 1

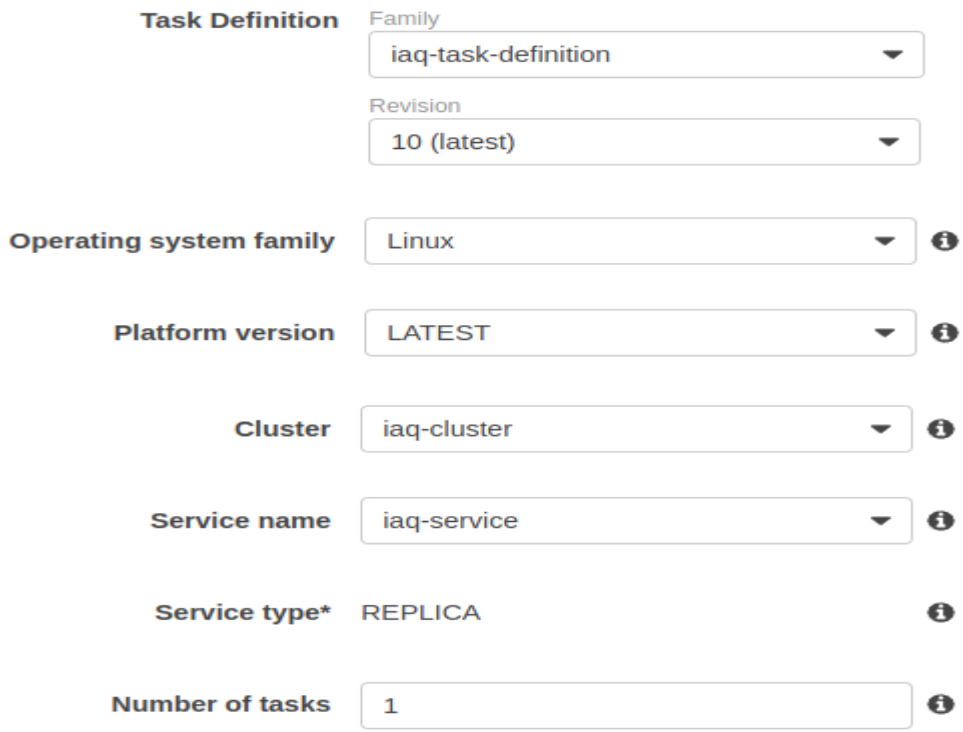
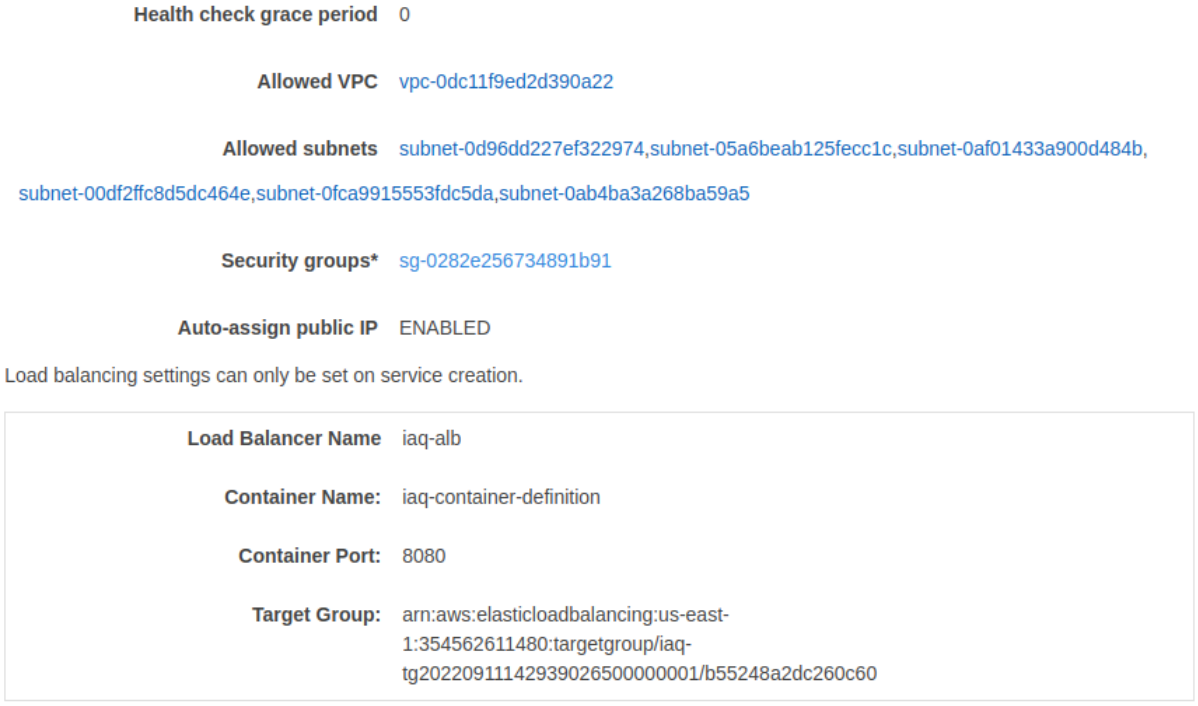


Figura 29
Atributos usados para la creación del servicio en ECS - parte 2

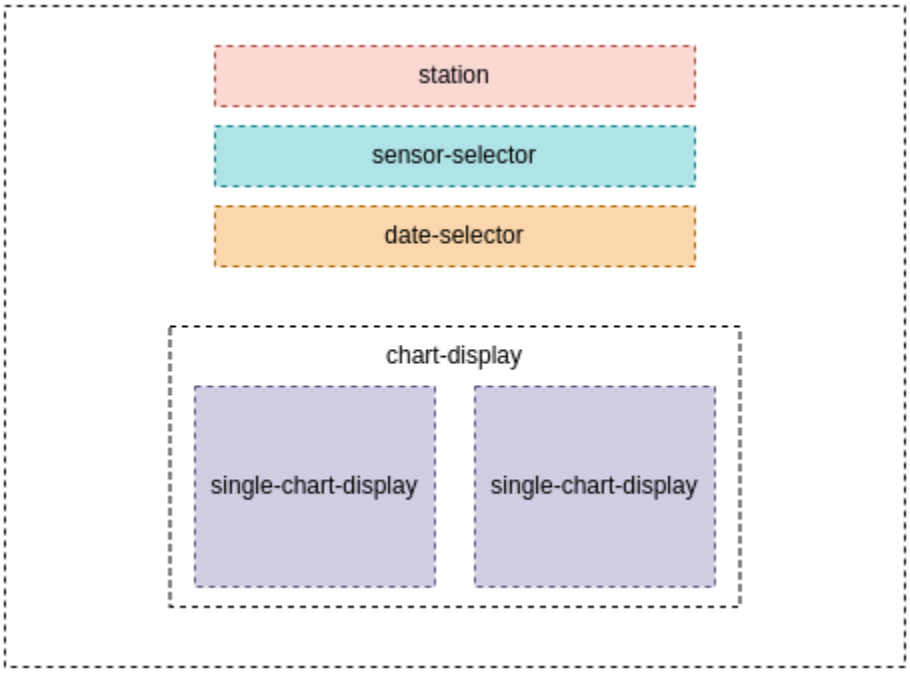


Una vez creado el servicio en ECS, pudimos invocar la API usando el DNS proveído por el balanceador de carga ya que este redirecciona el tráfico al servicio de ECS.

3.8 Creación del FrontEnd para la visualización de los datos

Para lograr la visualización de los datos desde una página web, se usó el framework Angular el cual nos facilitó la creación de componentes junto con la librería chart.js que permite la visualización de datos por medio de gráficos de línea. La figura 30 muestra un diagrama simplificado con los componentes principales que conforman el frontend

Figura 30
Componentes principales que conforman el frontend



El componente sensor-selector permite seleccionar cuales sensores se quieren visualizar. El componente date-selector es un calendario el cual permite seleccionar un rango de fecha para visualizar los datos históricos o permite seleccionar si se desean ver los datos en tiempo real. El componente single-chart-display se genera por cada sensor seleccionado y representa un gráfico de línea con los datos del sensor en el lapso de tiempo seleccionado.

Para lograr la visualización de los datos históricos desde el frontend, primero se debió instanciar un objeto de tipo Chart el cual representa un gráfico en el cual podemos graficar líneas de acuerdo con unos datos previamente cargados. El siguiente código muestra el proceso simplificado de cómo se pudo lograr esto.

```
// HTML
<div>
  <canvas id="chart"></canvas>
</div>

// JavaScript
const config = {
    type: 'line',
    data: data,
    options: options
};
const chart = new Chart(
    document.getElementById('chart'),
    config
);
```

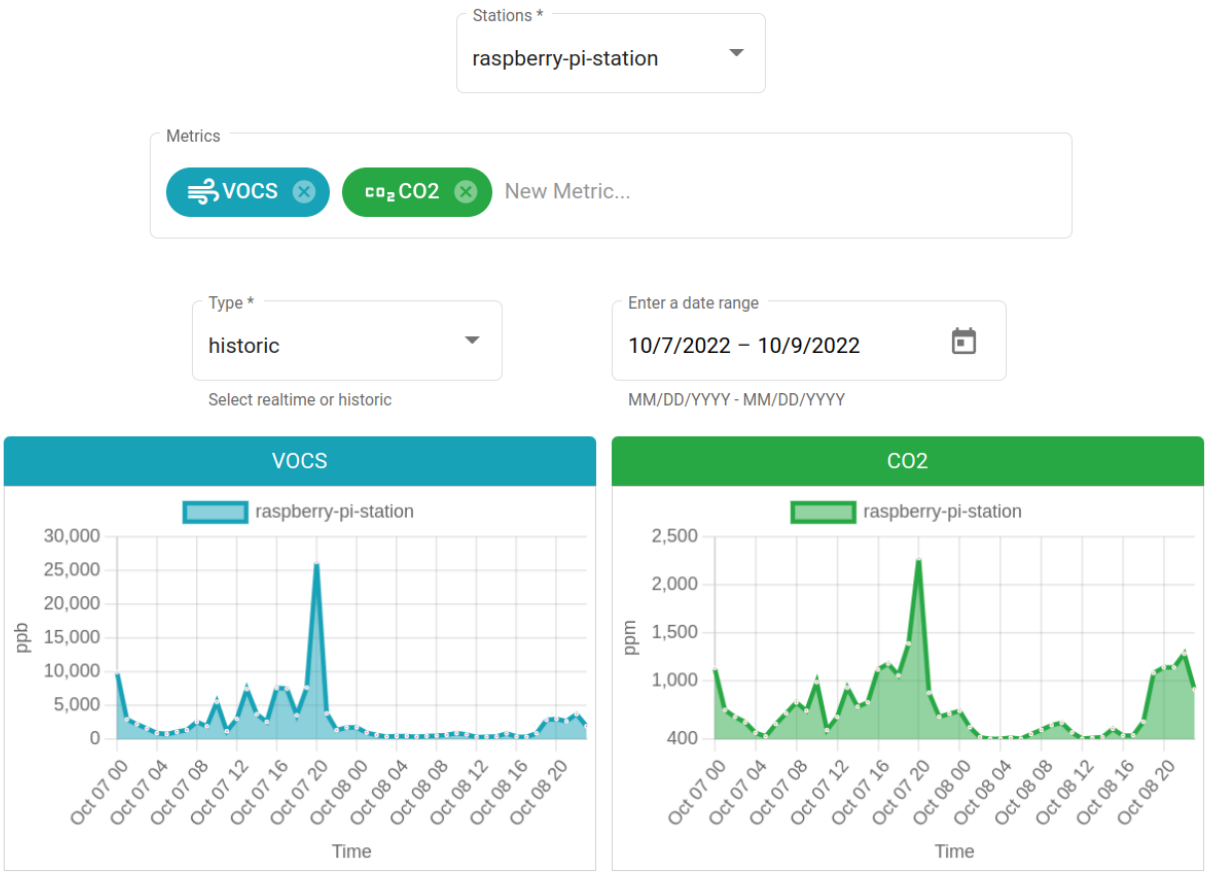
Un atributo por destacar es el objeto data el cual representa los puntos que se van a graficar. Estos datos se obtuvieron haciendo llamados HTTP a la API usando el rango de tiempo ingresado por el usuario y las métricas o sensores que quiere visualizar. El siguiente código muestra un ejemplo de cómo se realizó la petición HTTP usando el módulo HTTPClient de Angular.

```
let params = new HttpParams()
                      .set('start', start.toISOString())
                      .set('end', end.toISOString());
```

```
httpClient.get(url, {params:params}).subscribe((response) => {
      this.chart.datasets.push(response.data);
      this.chart.update();
});
```

La figura 31 muestra el frontend para la visualización de los datos para los sensores de CO2 y VOCS. Allí se puede observar la selección de los sensores de una lista, junto con el rango de fecha en la cual se quiere consultar los datos.

Figura 31
Visualización de datos históricos (CO2 y VOCs) desde el frontend



Para lograr la visualización en tiempo real desde el frontend fue necesario suscribir la aplicación a los mismos topics MQTT en AWS IoT Core usando el SDK de Amazon llamado

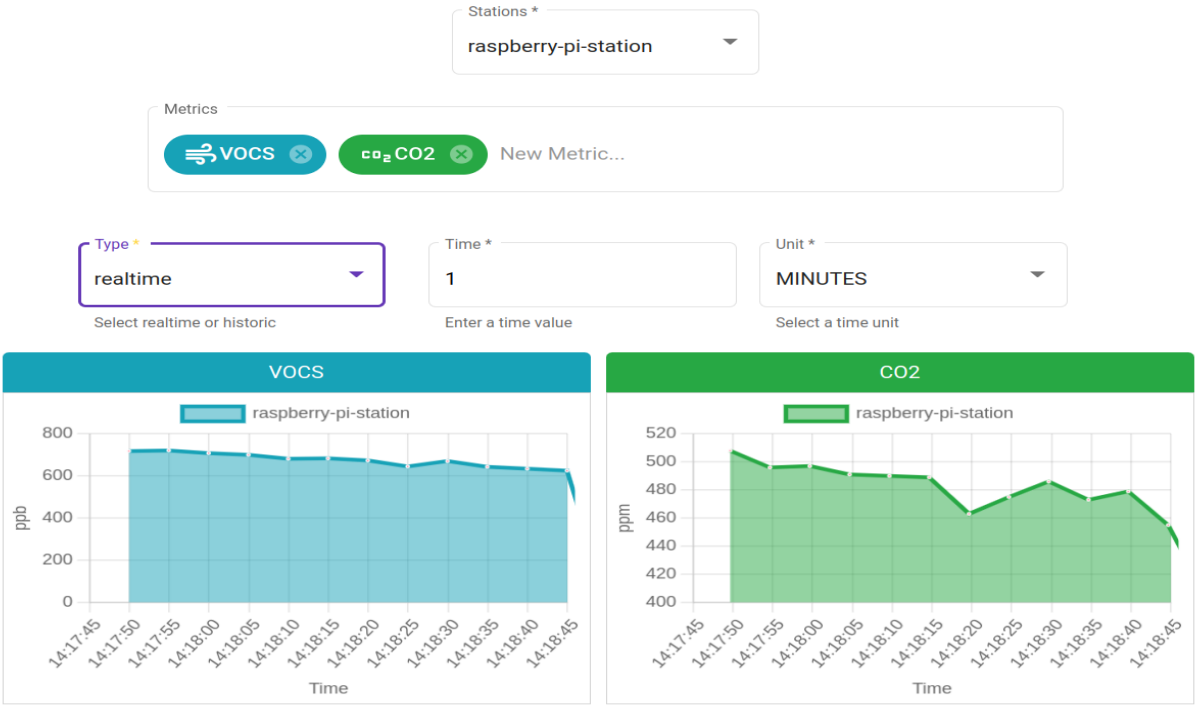
Amplify.js el cual provee funciones para interactuar con los servicios de AWS desde una aplicación escrita en JavaScript. Amplify nos permitió conectarnos a AWS IoT Core de una forma parecida a como se hizo desde la Raspberry Pi, solo que en este caso se usa JavaScript en vez de Python. El siguiente código muestra cómo se logra la suscripción usando esta librería.

```
const subscription = Amplify.PubSub.subscribe(topic);
subscription.subscribe(data => {
      this.chart.datasets.push(data);
      this.chart.update();
})
```

El objeto subscription nos permite ejecutar una función cada vez que se recibe un dato en el topic especificado. En este caso, agregamos el nuevo dato a la instancia de tipo Chart para que se muestre en tiempo real en el navegador web.

La figura 32 muestra el frontend para la visualización en tiempo real para los sensores de CO2 y VOCS. El tipo de selección que en este caso es REALTIME.

Figura 32
Visualización de datos en tiempo real (CO2 y VOCs) desde el frontend



El tiempo en este caso determina el lapso de retención de los datos anteriores recibidos en tiempo real. Por ejemplo, 1 minuto en este caso significa que la gráfica retiene los datos recibidos en el último minuto. Los datos que superen el minuto son eliminados de la gráfica y esta se va actualizando en tiempo real a medida que la estación va reportando datos nuevos.

3.8.1 Despliegue usando GitHub Pages

El despliegue del frontend se hizo usando Github Pages el cual nos permitió alojar el sitio web con contenido estático (HTML, CSS y Javascript) directamente desde un repositorio público alojado en Github. Github pages tiene un límite de 100GB de ancho de banda por mes haciéndolo muy adecuado para el despliegue de prototipos sin necesidad de pagar un servidor dedicado.

Para lograr esto, simplemente se instaló el plugin de angular llamado angular-cli-ghpages el cual nos permite hacer el despliegue directamente desde un proyecto basado en angular con un comando. El siguiente código muestra el comando para realizar el despliegue.

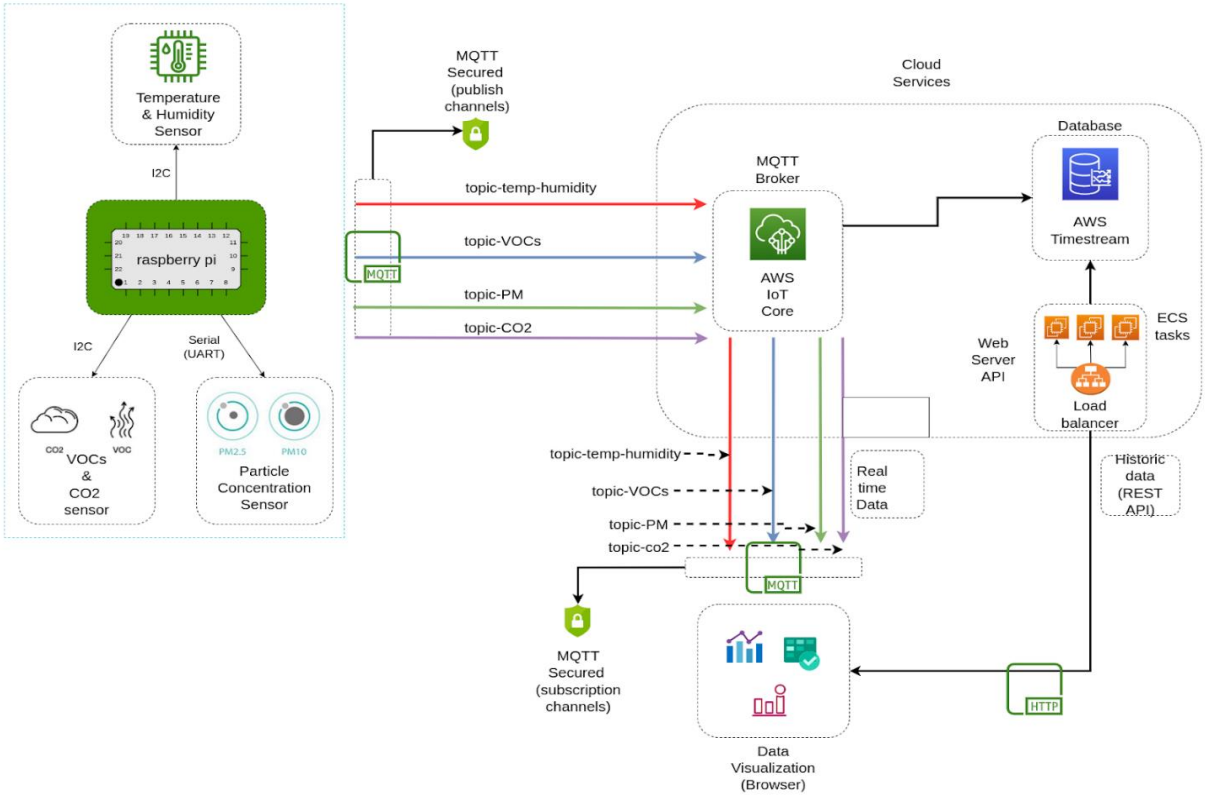
```
ng deploy --build-target=prod
```

Luego de ejecutar este comando, Github se encargó de desplegar nuestro código del frontend y el resultado es una URL desde la cual podemos acceder al frontend.

3.9 Arquitectura

La figura 33 muestra la arquitectura general de todo el sistema. La estación recolecta los datos de los sensores y los envía a AWS IoT core. Este envía los datos a AWS Timestream los cuales pueden ser consultados por medio de una API. El aplicativo web se conecta a AWS IoT para recibir los datos en tiempo real y a la API para consultar los datos históricos.

Figura 33
Arquitectura general del sistema de monitoreo remoto

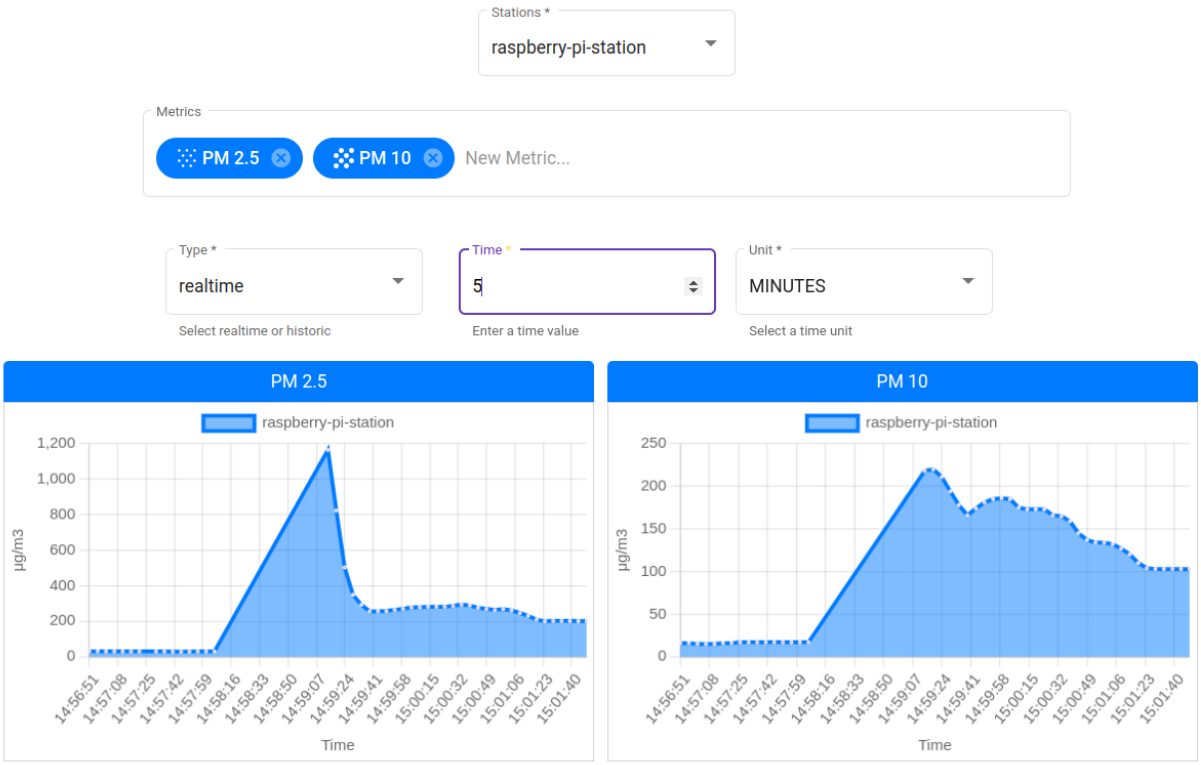


Si se desea expandir la red, simplemente registrar y conectar una estación con diferente id a la misma red.

4 Resultados

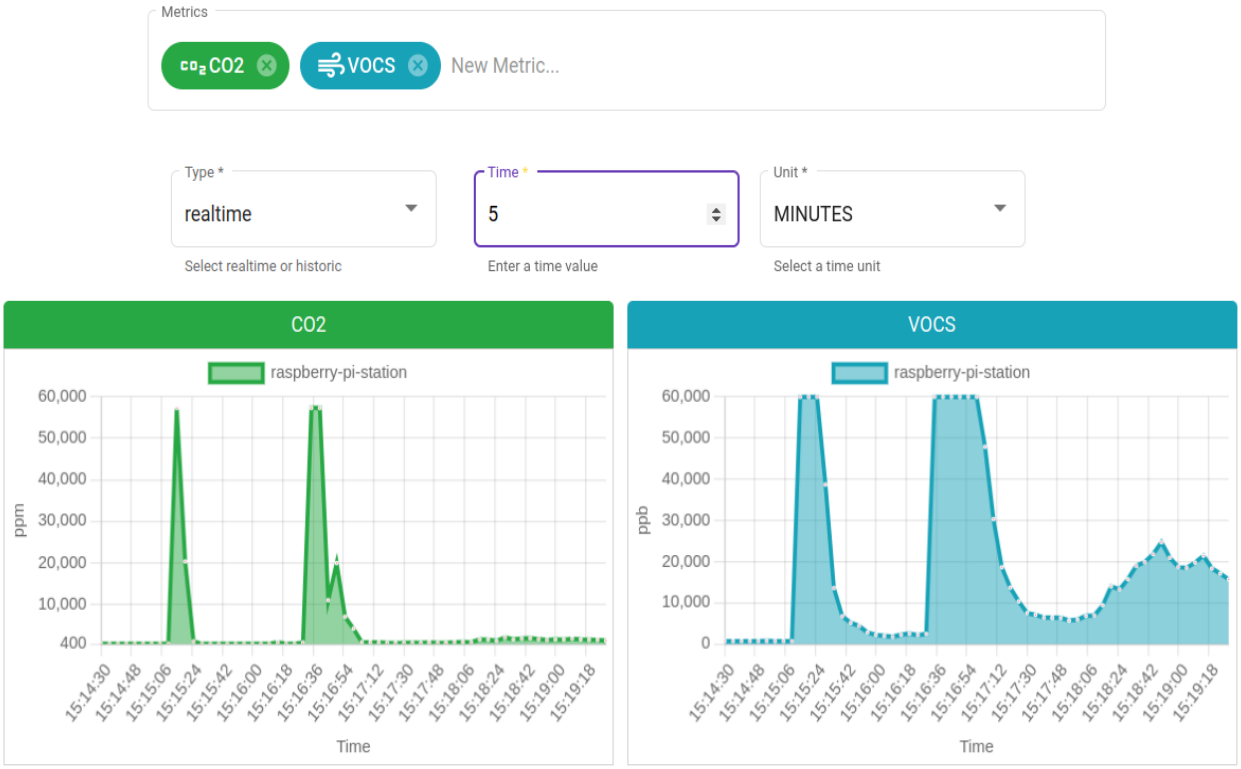
Para comprobar las medidas de material particulado, se sometió la estación al humo de estaño generado por medio de un soldador. Este humo ingresó por la cavidad de aire del sensor de material particulado y se observó cómo incrementó el valor en tiempo real desde la aplicación web. La figura 34 muestra las gráficas en tiempo real de PM2.5 Y PM10.

Figura 34
Gráficas de PM2.5 y PM10 usando humo de estaño



Para comprobar las medidas de CO2 y VOCS, se sometió la estación a gases generados usando desodorante en aerosol y se observó cómo incrementó el valor en tiempo real desde la aplicación web. La figura 35 muestra las gráficas en tiempo real de CO2 y VOCs

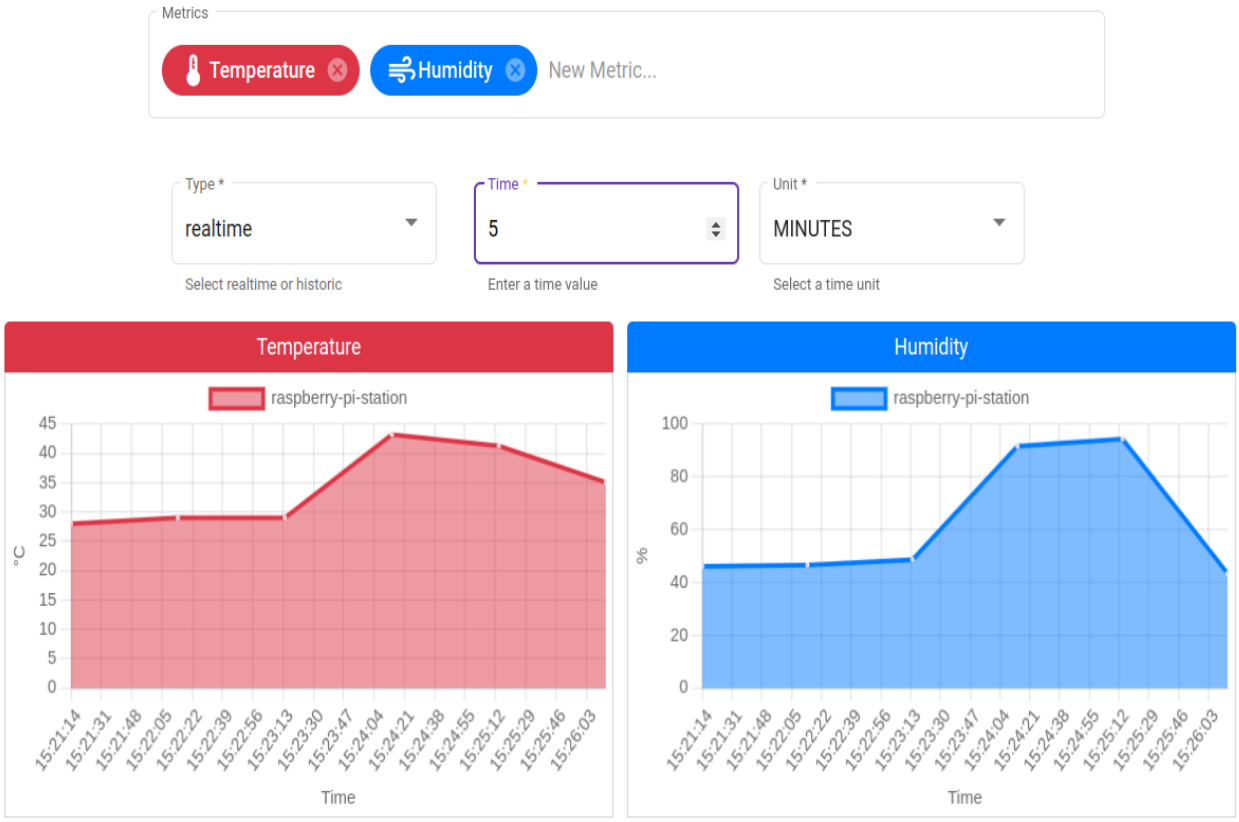
Figura 35
Gráficas de CO2 y VOCs usando desodorante en aerosol



Igualmente se comprobó el funcionamiento del sensor de temperatura y humedad usado vapor caliente. La figura 36 muestra las gráficas en tiempo real de temperatura y humedad.

Figura 36

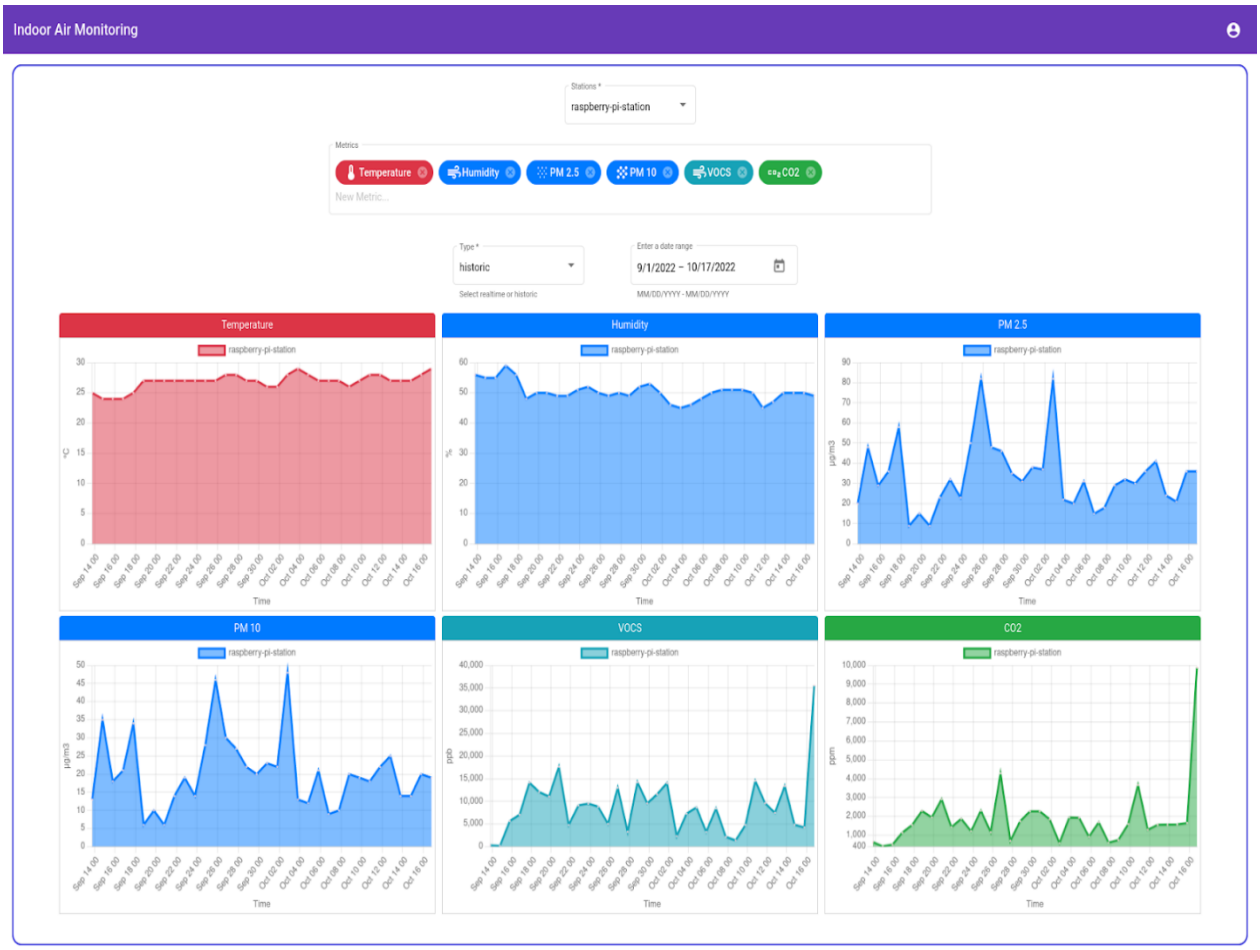
Gráficas de temperatura y humedad usando vapor de agua



La figura 37 muestra la vista del frontend con las gráficas de todos los sensores usando la opción de tiempo histórico en un periodo de un mes.

Figura 37

Gráficas de VOCs, PM2.5, PM10, CO2, temperatura y humedad en un periodo de un mes.



La figura 38 y 39 muestran la estación física luego de ser ensamblada.

Figura 38

Ensamblaje de la estación de monitoreo parte 1

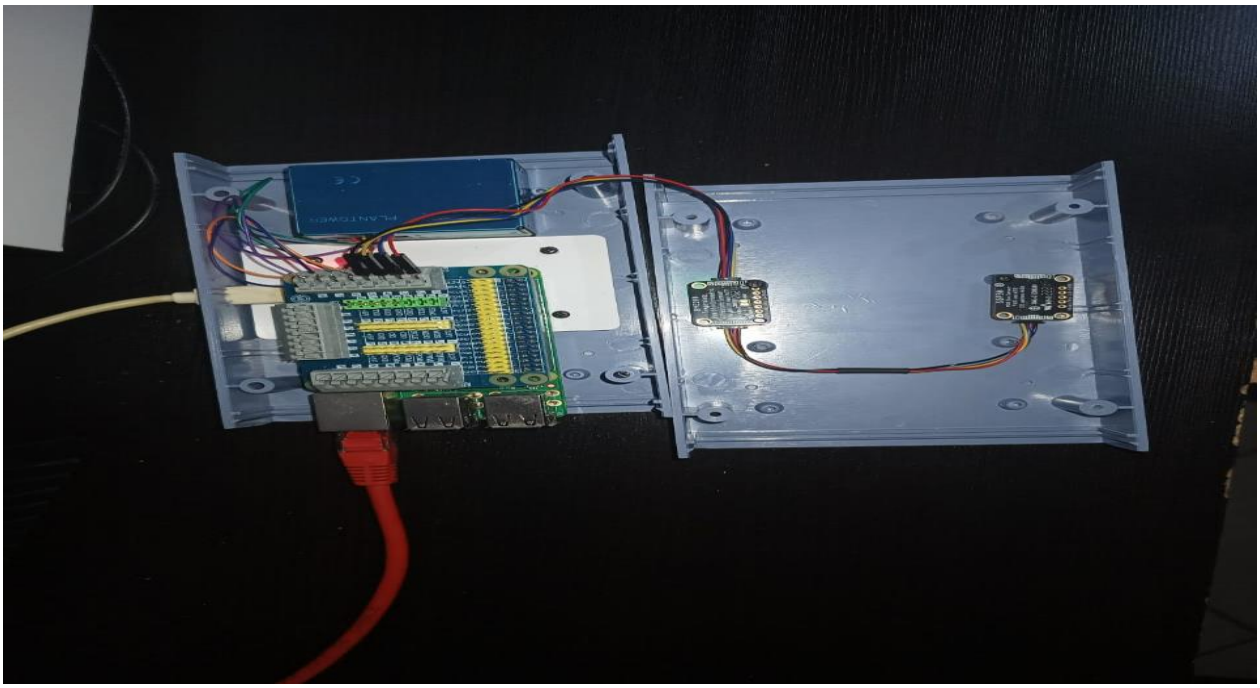


Figura 39

Ensamblaje de la estación de monitoreo parte 2

5 Conclusiones

La estación desarrollada proporcionó un funcionamiento adecuado en la recolección de los datos de CO2 y VOCs durante el funcionamiento normal y durante la fase de pruebas al ser sometida a gases como desodorante en aerosol y productos de limpieza.

En algunos casos fue posible apreciar picos por encima de 60000ppm y 60000ppb para las variables de CO2 y VOCs durante el funcionamiento normal de la estación (sin realizar pruebas sometiendo el sensor directamente a gases). Por lo tanto, se debe considerar el uso de un sensor con mayor rango.

La estación proporcionó un funcionamiento adecuado en la recolección de los datos de PM2.5 y PM10 durante el funcionamiento normal y durante la fase de pruebas al ser sometida a humo de estaño.

La estación proporcionó un funcionamiento adecuado en la recolección de los datos de temperatura y humedad durante el funcionamiento normal y durante la fase de pruebas al ser sometida a vapor de agua.

A pesar de que el envío de datos cada 5 segundos fue suficiente para visualizar los cambios en los sensores, se podría considerar aumentar el tiempo a 1 minuto para las variables de temperatura y humedad debido a que estas cambian lentamente a diferencia de las demás variables que pueden generar picos en periodos de unos pocos segundos. Este cambio ocasionaría una reducción de costos en AWS.

El aplicativo web y la API se mantuvieron funcionales en cuanto a los tiempos de repuesta durante el proceso de obtención de los datos en la base de datos y en graficar los datos en la interfaz de usuario. Aunque los tiempos de respuesta de la API varían dependiendo del rango de tiempo, estos se mantuvieron por debajo de los 4 segundos en la mayoría de los casos.

6 Referencias

Adafruit. (n.d.). Adafruit BME280 I2C or SPI Temperature Humidity Pressure Sensor. Adafruit Industries. https://www.adafruit.com/product/2652

Adafruit. (n.d.). Adafruit SGP30 Air Quality Sensor Breakout - VOC and eCO2. Adafruit Industries. https://www.adafruit.com/product/3709

Amazon Web Services. (n.d.). What Is Amazon Timestream? - Amazon Timestream. AWS Documentation. https://docs.aws.amazon.com/timestream/latest/developerguide/what-is-timestream.html

Amazon Web Services. (n.d.). What is AWS IoT? - AWS IoT Core. AWS Documentation. https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html

BOSCH. (2022, January). BME280 – Data sheet. BME280 Combined humidity and pressure sensor. https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf

Departamento Nacional de Planeación. (2017, May 7). Los costos en la salud asociados a la degradación ambiental en Colombia ascienden a $20,7 billones. Departamento Nacional de Planeación. https://www.dnp.gov.co/Paginas/Los-costos-en-la-salud-asociados-a-la-degradaci%C3%B3n-ambiental-en-Colombia-ascienden-a-$20,7-billones-.aspx

Greiner, T. (1991). Indoor Air Quality: Carbon Monoxide and Carbon Dioxide (AEN-125) - Department of Agricultural and Biosystems Engineering. Iowa State University Department of Agricultural and Biosystems Engineering. https://www.abe.iastate.edu/extension-and-outreach/indoor-air-quality-carbon-monoxide-and-carbon-dioxide-aen-125/

Ministerio de Ambiente y Desarrollo Sostenible. (n.d.). Contaminación Atmosférica En Colombia. Ministerio de Ambiente y Desarrollo Sostenible. https://www.minambiente.gov.co/asuntos-ambientales-sectorial-y-urbana/contaminacion-atmosferica/

Ministerio De Ambiente y Desarrollo Sostenible. (2017, November 1). Resolución 2254 de 2017. Por la cual se adopta la norma de calidad del aire ambiente y se dictan otras disposiciones. http://www.ideam.gov.co/documents/51310/527391/2.+Resoluci%C3%B3n+2254+de+2017+-+Niveles+Calidad+del+Aire..pdf/c22a285e-058e-42b6-aa88-2745fafad39f

OASIS. (march, 7 7). MQTT Version 5.0. MQTT Version 5.0. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

Sensirion. (2020, May). Indoor Air Quality Sensor for TVOC and CO2eq Measurements. Datasheet SGP30. https://sensirion.com/media/documents/984E0DD5/61644B8B/Sensirion_Gas_Sensors_Datasheet_SGP30.pdf

US Environmental Protection Agency. (2021, May 26). Particulate Matter (PM) Basics | US EPA. US Environmental Protection Agency. https://www.epa.gov/pm-pollution/particulate-matter-pm-basics

US Environmental Protection Agency. (2021, September 24). Volatile Organic Compounds' Impact on Indoor Air Quality | US EPA. US Environmental Protection Agency. https://www.epa.gov/indoor-air-quality-iaq/volatile-organic-compounds-impact-indoor-air-quality

Yong, Z. (2016, June 1). PMS5003 series data manual. Digital universal particle concentration sensor. https://www.aqmd.gov/docs/default-source/aq-spec/resources-page/plantower-pms5003-manual_v2-3.pdf