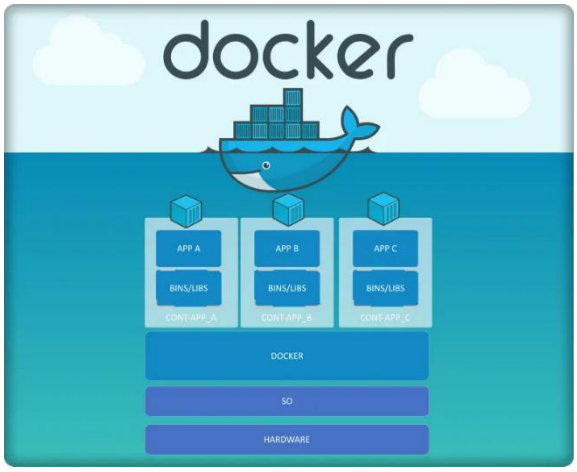
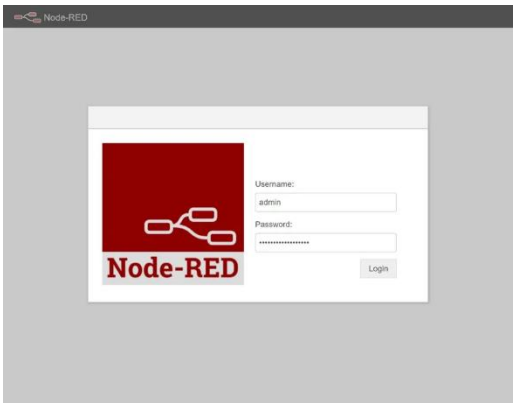
UNIVERSIDAD TECNOLOGICA NACIONAL
Facultad Regional Bahía Blanca

Proyecto Final : “Capstone”

Año: 2025





Tecnicatura Universitaria en Programación

Cátedra “Arquitectura y Sistemas Operativos”

Profesor: Mateo Menvielle

Profesor: Mateo Benjamín Barrera

Descripción del Proyecto

Este informe describe el desarrollo de un sistema basado en Node-RED que consume datos desde una API REST pública, los procesa y visualiza en tiempo real a través de widgets en un dashboard. El sistema se ejecuta dentro de un contenedor Docker.

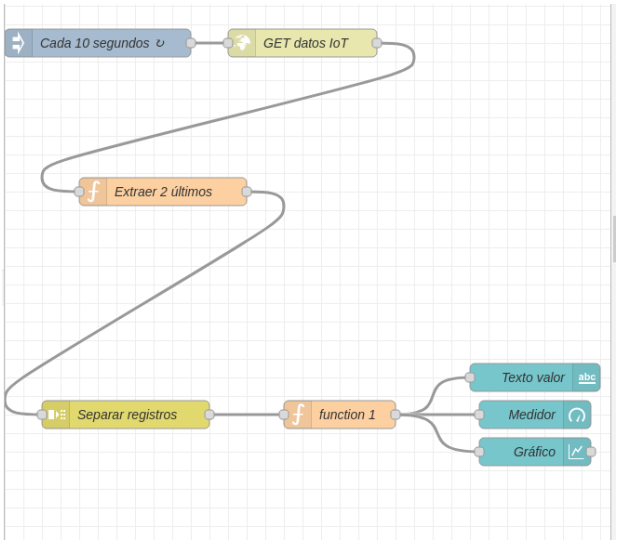
Utilizamos la Maquina Virtual (Virtual Box) para hacer el proyecto desde Linux(Ubuntu) ; creamos el flujo, luego desde el terminal con comandos vamos realizando los puntos pedidos por el profesor.

1. Ejemplo del JSON recibido mediante el DEBUG


```
[
{
"id": "abc001",
"value": 23.7,
"timestamp": "2025-07-17T19:00:00Z"
},
{
"id": "abc002",
"value": 22.8,
"timestamp": "2025-07-17T18:59:50Z"
}
]
```


2. Descripción del flujo implementado

El flujo de Node-RED inicia con un nodo 'inject' que ejecuta una consulta cada 10 segundos. Luego, un nodo 'http request' accede a los datos de la API. Un nodo 'function' ordena los registros por fecha y extrae los dos más recientes. Estos se dividen con un nodo 'split', y un nodo 'function' adicional extrae el valor numérico de cada registro. Finalmente, se visualizan en widgets 'ui_text', 'ui_gauge' y 'ui_chart' en el dashboard.



Explicaciones técnicas

Docker es una plataforma de virtualización ligera que permite empaquetar aplicaciones junto con todas sus dependencias en **contenedores**.
Un **contenedor** es una instancia ejecutable de una imagen de Docker, que incluye código, bibliotecas y configuraciones necesarias para que una aplicación funcione igual en cualquier entorno.
Ventajas técnicas:

- Aislamiento de aplicaciones.
- Portabilidad entre distintos sistemas operativos.
- Despliegues consistentes (ambientes iguales en desarrollo, testing y producción).
- Escalabilidad fácil usando orquestadores como Docker Swarm o Kubernetes.

🌐 APIs (Interfaces de Programación de Aplicaciones)

Una **API** es un conjunto de reglas que permite que dos programas se comuniquen entre sí. Generalmente, las APIs modernas son APIs **REST** que utilizan **HTTP** para intercambiar datos en formato **JSON**.
Componentes clave de una API REST:

- **GET**: obtener datos.
- **POST**: crear datos.
- **PUT/PATCH**: actualizar datos.
- **DELETE**: eliminar datos.

Ejemplo práctico:
Un sistema IoT puede usar una API para enviar datos del sensor a un servidor web o base de datos.

🌐 🔌 IoT (Internet of Things / Internet de las Cosas)

IoT se refiere a una red de dispositivos físicos ("cosas") conectados a internet que pueden recopilar e intercambiar datos.
Ejemplos comunes:

- Sensores de temperatura.
- Cámaras de vigilancia.
- Dispositivos domóticos (luces, cerraduras inteligentes, etc.).

Técnicamente incluye:

- Sensores/actuadores.
- Protocolos de comunicación (MQTT, HTTP, CoAP).
- Plataformas en la nube para visualización, almacenamiento y análisis.
- Gateways (como una Raspberry Pi o un contenedor Docker con Node-RED).

🔴 Node-RED

Node-RED es una herramienta de programación visual para conectar dispositivos, APIs y servicios en línea de forma sencilla, usando "nodos" que se arrastran y conectan entre sí.
Características técnicas:

- Basado en Node.js.
- Interfaz web para diseño de flujos.
- Integración fácil con MQTT, HTTP, bases de datos, APIs REST, etc.
- Ideal para proyectos IoT, automatización industrial, prototipado rápido.

Archivo docker-compose.yml – Definición de contenedores
docker-compose.yml permite definir y ejecutar múltiples contenedores Docker con una sola configuración.
Ejemplo típico con Node-RED e InfluxDB:

```yaml
version: '3.8'

services:
  nodered:
    image: nodered/node-red
    container_name: nodered
    ports:
      - "1880:1880"
    volumes:
      - ./nodered_data:/data
    restart: always
    networks:
      - iot_net

  influxdb:
    image: influxdb:latest
    container_name: influxdb
    ports:
      - "8086:8086"
    volumes:
      - ./influxdb_data:/var/lib/influxdb
    restart: always
    networks:
      - iot_net

networks:
  iot_net:
    driver: bridge
```

Explicación técnica:

- services: lista de contenedores a crear.
- image: nombre de la imagen que se usará para el contenedor.
- container_name: nombre del contenedor.
- ports: mapea los puertos del contenedor a los del host (host:container).
- volumes: persiste datos en el sistema host (evita que se borren al reiniciar).
- restart: política de reinicio (always, on-failure, etc.).
- networks: permite la comunicación interna entre contenedores en una red privada.

Mediante el endpoint público proporcionado por el profesor:
https://callback-iot.up.railway.app/data
Realizamos la Practica en Node Red:

🔁 **Descripción detallada del flujo Node-RED**

🕒 **1. Nodo: "Cada 10 segundos" (Inject)**

- Este nodo es un disparador automático que ejecuta el flujo **cada 10 segundos**.
- Simula una tarea programada que inicia el proceso de consulta de datos IoT periódicamente.

🌐 **2. Nodo: "GET datos IoT" (HTTP request)**

- Realiza una **solicitud HTTP GET** a una API o servidor donde están alojados los datos de los sensores IoT.
- Este nodo espera una respuesta tipo JSON (probablemente una lista de registros).

🧩 **3. Nodo: "Extraer 2 últimos" (Function)**

- Este nodo **filtra o recorta** la respuesta obtenida anteriormente.
- Se queda con **los dos registros más recientes**.
- Puede usar una lógica similar a:
- msg.payload = msg.payload.slice(-2);
- return msg;

🔄 **4. Nodo: "Separar registros" (Split)**

- Toma la lista de los dos registros y **la divide en dos mensajes individuales** (uno por cada registro).
- Así se pueden procesar y visualizar por separado.

🧠 **5. Nodo: "function 1" (Function)**

- Esta función probablemente **extrae el valor numérico** de interés (por ejemplo, temperatura, humedad, etc.) del objeto JSON de cada mensaje.
- También puede incluir lógica para convertir unidades, calcular promedio, etc.
- msg.payload = msg.payload.valorSensor;
- return msg;

📊 **6. Nodos de visualización:**

Estos tres nodos muestran los datos procesados:

- **Texto valor (ui_text)**: Muestra el valor como texto plano.
- **Medidor (ui_gauge)**: Representa el valor en un medidor circular, útil para monitoreo.
- **Gráfico (ui_chart)**: Muestra la evolución del dato en el tiempo (acumulando registros).

🧭 **Flujo general resumido:**

[Trigger cada 10s] ➜ [HTTP GET datos IoT] ➜ [Extraer 2 últimos registros] ➜
➜ [Separar en mensajes individuales] ➜ [Extraer dato clave] ➜ [Visualización]
➜ Texto / Medidor / Gráfico

✅ **Ventajas de este diseño:**

- Actualización automática periódica.
- Visualización en tiempo real en múltiples formatos.
- Separación clara entre obtención, transformación y visualización de datos.

PASOS PARA CORRER DOCKER-COMPOSE

1. Abrir la terminal

2. Navegar a la carpeta del proyecto donde está el archivo docker-compose.yml
   Ejemplo:
   cd ~/proyecto-nodered

3. Ejecutar el comando para levantar los servicios en segundo plano:
   docker compose up -d

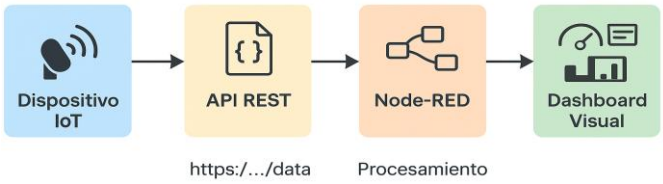
   Si hay problemas de permisos, usar:
   sudo docker compose up -d

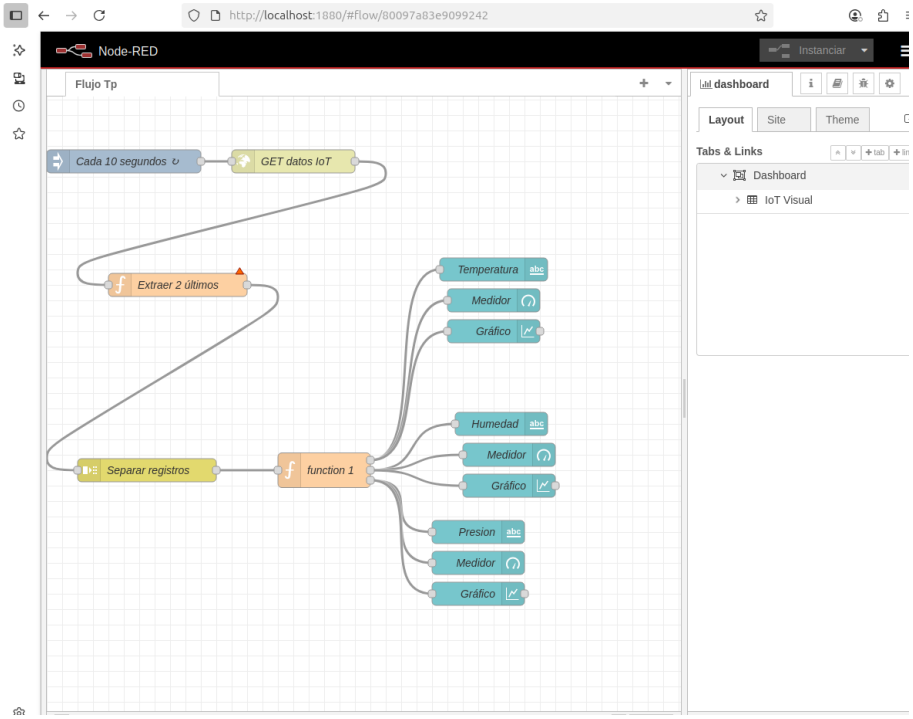
4. Verificar que el contenedor esté corriendo:
   docker compose ps

5. (Opcional) Ver logs en tiempo real:
   docker compose logs -f

6. Para detener y eliminar los contenedores:
   docker compose down

Imagen del flujo de Red y del diagrama en bloques; luego a continuación seguimos con las imágenes de los nodos y sus configuraciones:





Editar nodo inject

Eliminar | Cancelar | Hecho

Propiedades

Nombre: Cada 10 segundos

+ añadir | inyectar ahora

Inyectar una vez después de 1 segundos, entonces

Repetir: intervalo

cada 10 segundos

Habilitado

Editar nodo http request

Eliminar | Cancelar | Hecho

Propiedades

Método: GET

URL: https://callback-iot.up.railway.app/data

Carga: Enviar como cuerpo de la solicitud

- [ ] Habilitar conexión segura (SSL/TLS)
- [ ] Usar autenticación
- [ ] Habilitar conexión activa (keep-alive)
- [ ] Usar proxy
- [ ] Enviar solo respuestas que no sean 2xx al nodo Catch
- [ ] Deshabilitar el análisis estricto de HTTP

Devolver: un objeto JSON analizado

Tip: If the JSON parse fails the fetched string is returned as-is.

Encabezados

Editar nodo function

Eliminar | Cancelar | Hecho

Propiedades

Nombre: Extraer 2 últimos

module msg
var msg: NodeMessage
@type — {NodeMessage} the msg object

Configuración | Al final

```
let data = msg.payload;

// Si el payload viene como string, lo parseamos
if (typeof data === "string") {
    try {
        data = JSON.parse(data);
    } catch (err) {
        node.error("Error al parsear JSON", err);
        return null;
    }
}

// Verificamos que sea un array
if (!Array.isArray(data)) {
    node.error("La respuesta no es un array");
    return null;
}

// Ordenamos por timestamp descendente
data.sort((a, b) => new Date(b.timestamp) - new Date(a.

// Tomamos los 2 más recientes
msg.payload = data.slice(0, 2);

return msg;
```

Habilitado

Editar nodo split

Eliminar | Cancelar | Hecho

Propiedades

Nombre: Separar registros

Dividir el: msg. payload

Texto / Buffer

Dividir usando: \n

- [ ] Manejar como un flujo de mensajes

Array

Dividir usando: Longitud fija de 1

Objeto

Enviar un mensaje para cada par clave/valor

- [ ] Copiar clave a msg.

Habilitado

Editar nodo function

Eliminar | Cancelar | Hecho

Propiedades

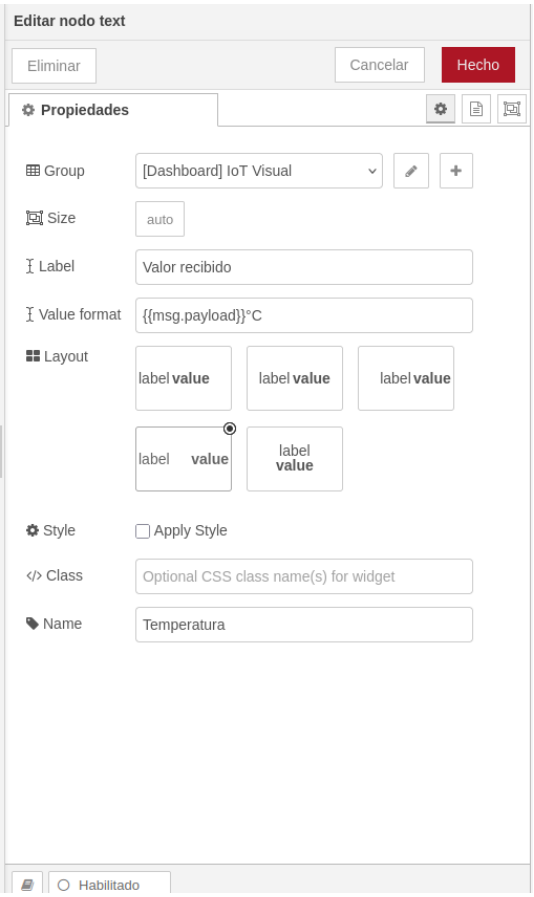
Nombre: function 1

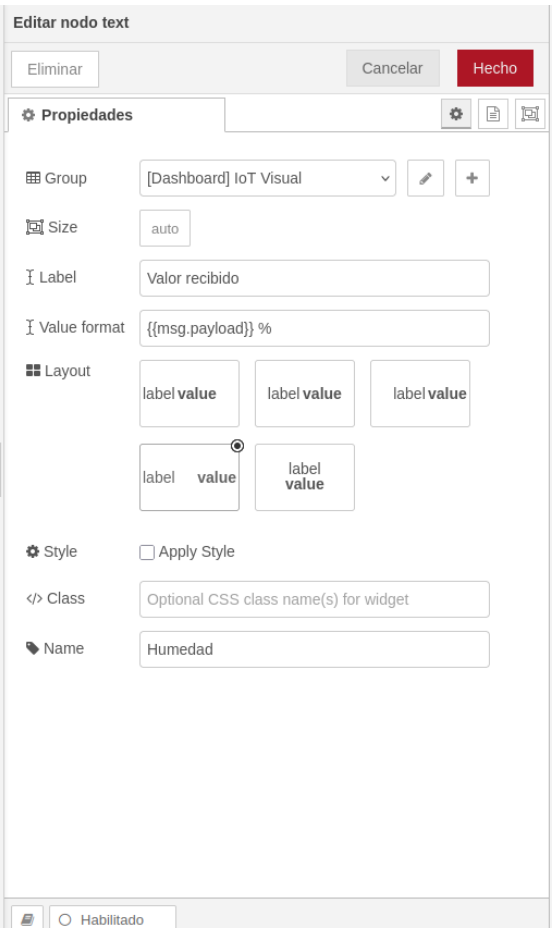
Configuración | Al inicio | **En mensaje** | Al final

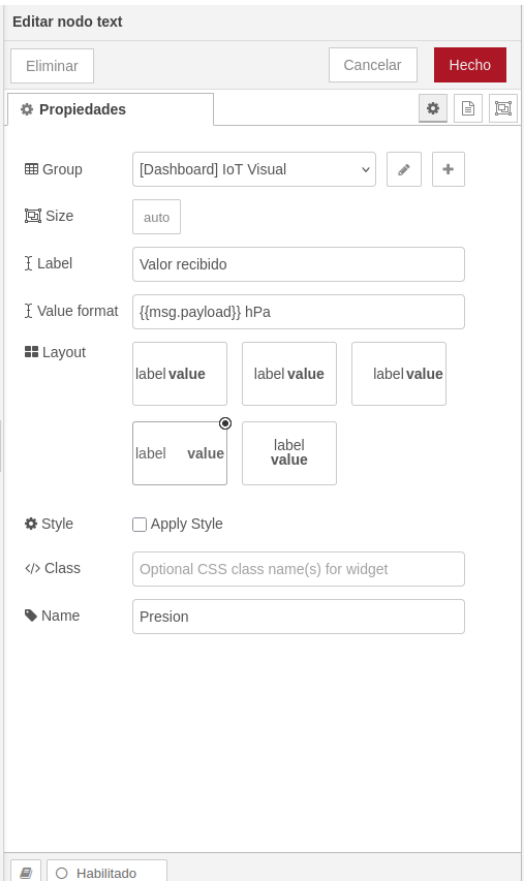
```
// Extraemos los datos del último objeto recibido
let t = parseFloat(msg.payload.temperature);
let h = parseFloat(msg.payload.humidity);
let p = parseFloat(msg.payload.pressure);

// Enviamos 3 mensajes distintos para cada tipo de dato
return [
    { payload: t, topic: "temperatura" },
    { payload: h, topic: "humedad" },
    { payload: p, topic: "presion" }
];
```

Habilitado

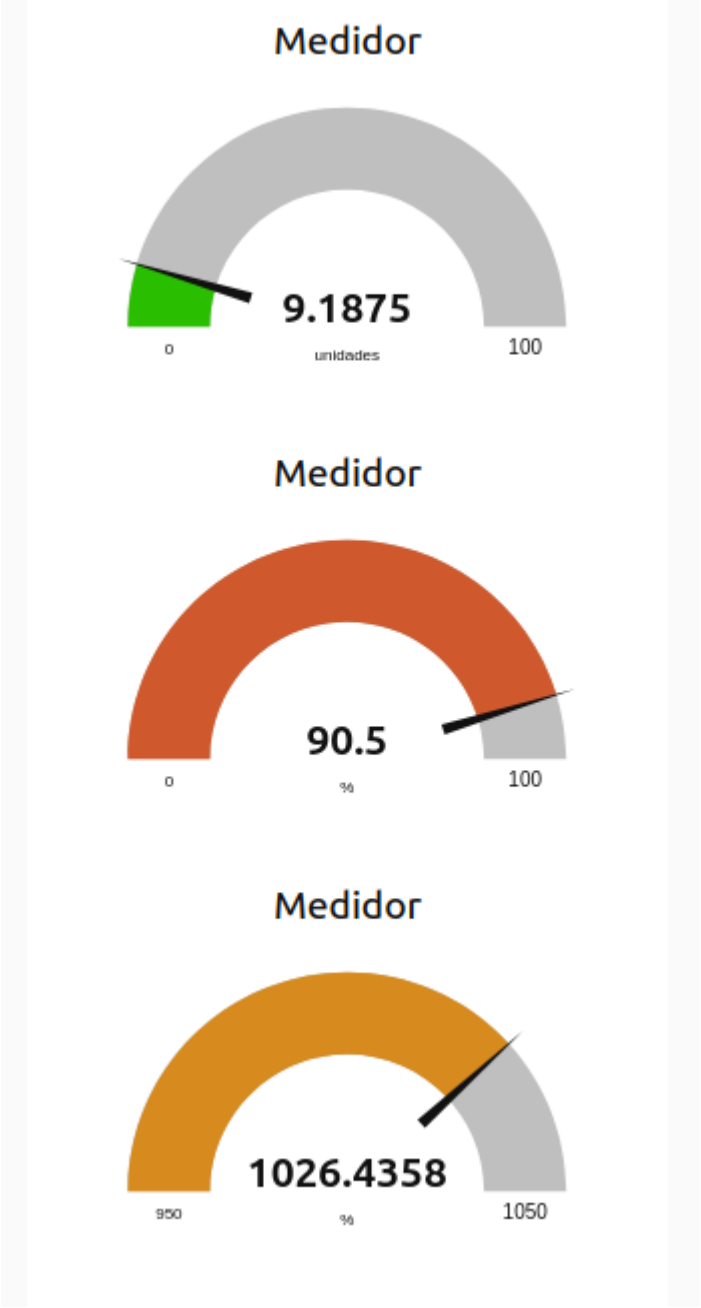
Configuración de los widgets como Gauge, Chart, Text. En el diagrama vemos 3 salidas ya que una de las funciones la configuramos para visualizar los datos de la temperatura, presión y humedad.







Luego una vez finalizado la configuración, accedemos a las visualizaciones de los dashboard. Tanto de los valores recibidos, como el del medidor y historial de los registros.



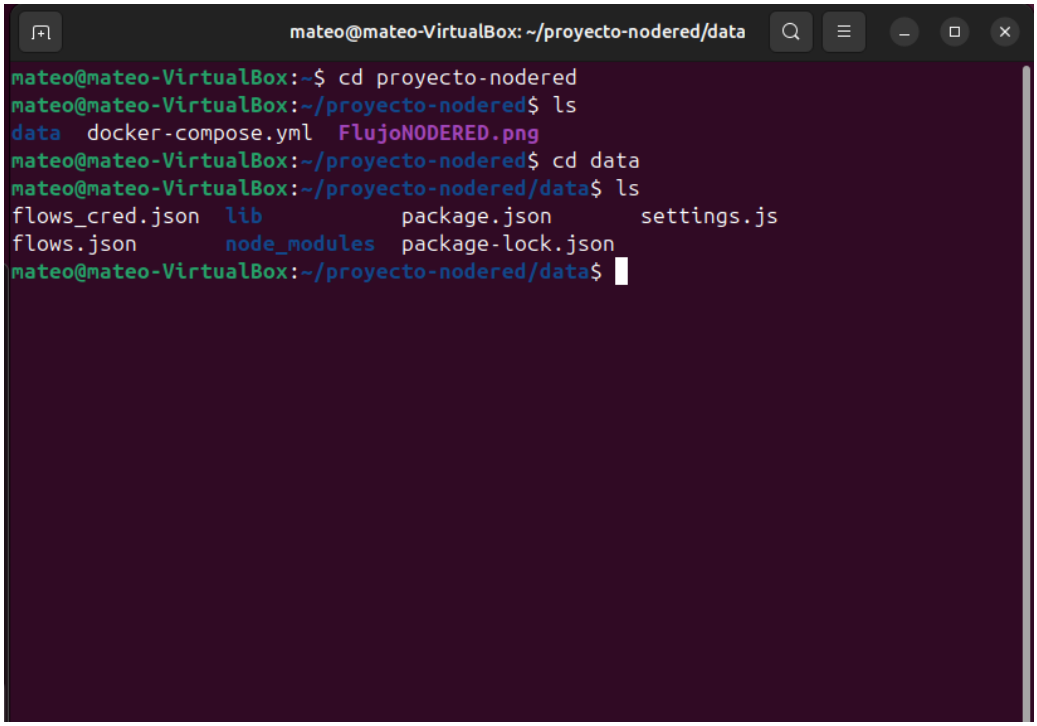
Reflexión:

Fue una práctica muy entretenida y buen proyecto para conocer mas sobre como se conectan las cosas o cómo funcionan. Lo más fácil fue lo de programar y hacer el diagrama del Node Red; después de hacer lo mencionado, se empezó a complicar un poco para dockerizar y usar desde Ubuntu (algo que mucho no usamos el día a día), pero pudimos seguir para adelante y avanzar, conocer funcionalidades nuevas desde Ubuntu con el terminal y comandos. A la hora de armar el docker-compose.yml; surgieron muchos errores (no saber como hacerlo) y recurri a la ayuda de IA. Logre hacerlo funcionar y seguir con el Proyecto.

Luego, también tuve errores en la hora de visualizar los dashboard , asi que tuve que implementar una funcion para poder ver resultados; y como se ve en el diagrama también separamos para poder visualizar mejor.

En síntesis:

Durante el desarrollo del proyecto aprendí a integrar múltiples tecnologías como Node-RED, Docker y APIs REST para construir una solución IoT completa. La parte más desafiante fue entender el flujo de datos JSON y cómo visualizar varios valores en tiempo real. Me resultó útil usar Docker para asegurar que el sistema funcione igual en distintos entornos. Este proyecto me permitió reforzar conocimientos prácticos sobre virtualización, automatización y visualización de datos.



```
mateo@mateo-VirtualBox: ~/proyecto-nodered
mateo@mateo-VirtualBox:~$ cd proyecto-nodered
mateo@mateo-VirtualBox:~/proyecto-nodered$ nano docker-compose.yml
mateo@mateo-VirtualBox:~/proyecto-nodered$ docker-compose up -d
[+] Running 1/1
 ✔ Container nodered  Started                                        1.1s
mateo@mateo-VirtualBox:~/proyecto-nodered$
```

```
mateo@mateo-VirtualBox: ~/proyecto-nodered
  GNU nano 7.2                    docker-compose.yml
version: '3.8'

services:
  nodered:
    image: nodered/node-red
    container_name: nodered
    ports:
      - "1880:1880"
    volumes:
      - ./nodered_data:/data
    restart: always

                          [ 12 líneas escritas ]
^G Ayuda      ^O Guardar    ^W Buscar     ^K Cortar     ^T Ejecutar   ^C Ubicación
^X Salir      ^R Leer fich. ^\ Reemplazar ^U Pegar      ^J Justificar ^/ Ir a línea
```

Fuentes : Inteligencia Artificial y teoría dada por el docente