PROYECTO CAPSTONE IOT

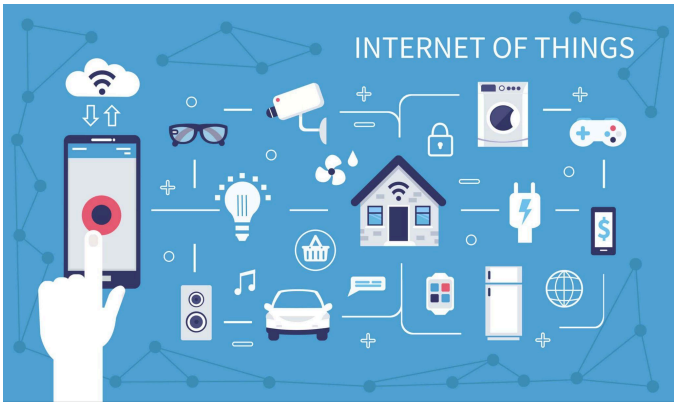
Nombre: **Homero Colombo**
Legajo: **23086**
Materia: Arquitectura y Sistemas Operativos
Facultad Regional Bahía Blanca, UTN
Tecnicatura Universitaria en Programación
10/07/2025
Profesor: **Mateo Menvielle**

Resumen

El presente proyecto tiene como objetivo diseñar, implementar y contenerizar una solución basada en Node-RED para el monitoreo y visualización de datos provenientes de un dispositivo IoT simulado, accediendo a través de una API pública. La aplicación se ejecuta en un entorno Linux (Ubuntu) dentro de una máquina virtual, y está completamente orquestada mediante Docker y Docker Compose, lo que garantiza portabilidad, automatización del despliegue y aislamiento del entorno de ejecución.

Mediante el uso de flujos de procesamiento visual, la solución realiza consultas periódicas a la API, transforma los datos JSON recibidos y los visualiza en tiempo real en un dashboard accesible vía navegador web. Los datos son filtrados, ordenados y presentados utilizando nodos funcionales y widgets de interfaz como `ui_gauge` y `ui_template`. Todo el flujo y la persistencia de la configuración se gestionan mediante volúmenes montados en el contenedor, definidos en el archivo `docker-compose.yml`.

Este proyecto integra conceptos fundamentales de la materia, como virtualización, planificación de procesos, arquitectura cliente-servidor, comunicación entre servicios, y gestión de recursos a través de contenedores, demostrando la aplicación práctica de estos conocimientos en el desarrollo de soluciones modernas y escalables.

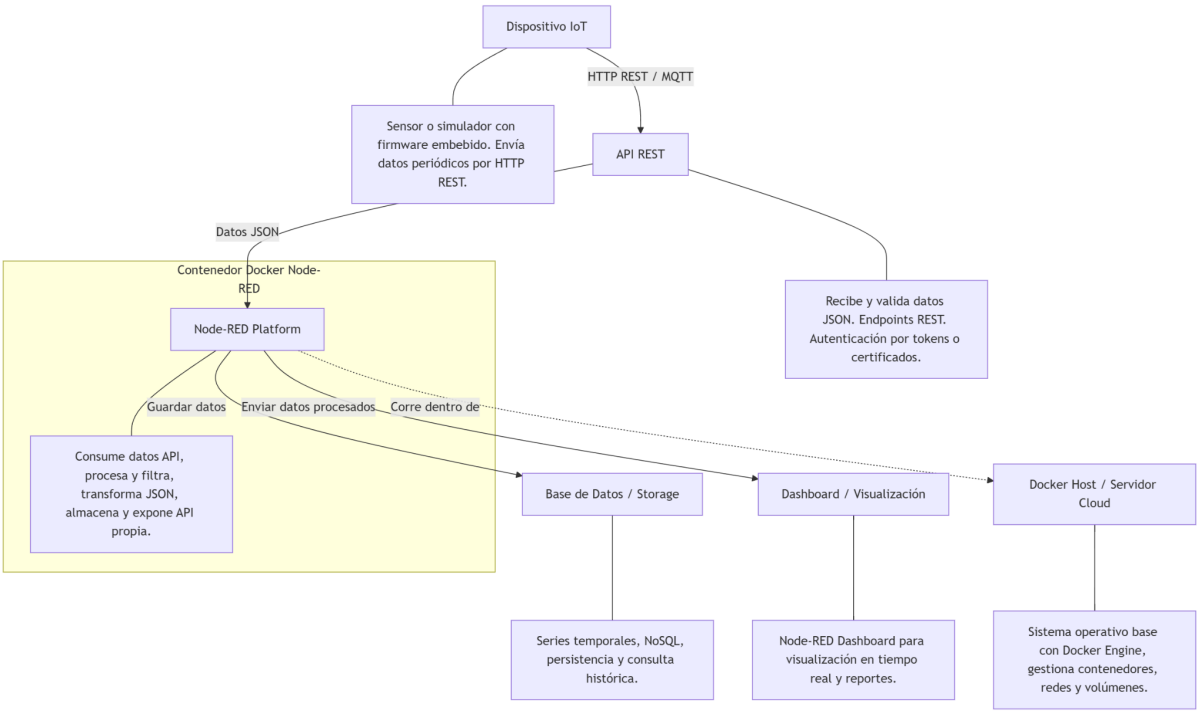


2. Introducción

El presente informe describe el desarrollo de un proyecto práctico cuya finalidad fue contenerizar una solución funcional utilizando Node-RED, una herramienta de programación visual orientada a flujos de datos. La propuesta consistió en consumir datos de un dispositivo IoT simulado mediante una API REST pública, procesarlos y visualizarlos en tiempo real dentro de un entorno contenerizado.

Para lograrlo, se utilizó Docker como plataforma de virtualización ligera, y Docker Compose como herramienta de orquestación, lo cual permitió desplegar toda la solución con un único archivo de configuración. El sistema se ejecutó en una máquina virtual con Ubuntu y fue construido para simular un entorno realista de integración entre servicios, similar a lo que se encuentra en aplicaciones modernas que combinan arquitectura de sistemas operativos con desarrollo de software.

Durante el proyecto se trabajó con conceptos claves como el aislamiento de entornos, persistencia de datos mediante volúmenes, servicios expuestos por puertos, consumo de APIs externas, y visualización de información mediante dashboards. Todo esto fue aplicado de manera integrada, poniendo en práctica conocimientos teóricos adquiridos en la materia Arquitectura y Sistemas Operativos.



Resumen técnico del sistema IoT

1. Dispositivo IoT / Simulador:
   Es el origen de los datos. Contiene sensores físicos o simulados que, mediante firmware embebido, generan y envían información en tiempo real a través de protocolos como HTTP REST.

2. API REST:
   Actúa como puerta de entrada. Recibe las mediciones del dispositivo, valida y estandariza los datos (en JSON), y puede aplicar autenticación. Funciona como interfaz entre el dispositivo y el sistema de procesamiento.

3. Node-RED (dentro de Docker):
   Plataforma de integración visual que recibe los datos desde la API o broker, los procesa mediante flujos (nodos), y los dirige hacia almacenamiento o visualización. Está contenerizado en Docker para garantizar portabilidad y aislamiento del entorno.

4. Docker Host / Servidor:
   Es la máquina (local o en la nube) que ejecuta Docker y, por tanto, el entorno de Node-RED. Gestiona los recursos del contenedor y permite escalar o replicar el sistema fácilmente.

5. Base de datos / Almacenamiento:
   Almacena los datos procesados de forma estructurada. Puede ser una base de series temporales (como InfluxDB) o documentos NoSQL (como MongoDB), permitiendo análisis posterior o explotación histórica.

6. Dashboard / Visualización:
   Muestra los datos en tiempo real en forma de gráficos, indicadores o tablas. Puede ser generado con Node-RED Dashboard o herramientas externas como Grafana, y es accesible desde un navegador web.

Conceptos Teóricos:

Docker

Docker es una plataforma de **contenedorización** que permite empaquetar aplicaciones y sus dependencias en un entorno aislado llamado **contenedor**. A diferencia de las máquinas virtuales tradicionales que incluyen todo un sistema

operativo completo, los contenedores comparten el kernel del sistema operativo del host, lo que hace que sean ligeros y eficientes en recursos.

- **Arquitectura**: Docker funciona sobre un modelo cliente-servidor. El **Docker Daemon** (dockerd) corre en segundo plano, administra la creación, ejecución y supervisión de contenedores. El cliente Docker (CLI) envía comandos al daemon mediante la API REST sobre UNIX sockets o TCP.

- **Imágenes**: Son plantillas inmutables basadas en capas de sistema de archivos unionfs (por ejemplo OverlayFS). Cada instrucción en un Dockerfile genera una nueva capa, que es una diferencia incremental respecto a la capa anterior. Las imágenes son almacenadas en repositorios (Docker Hub o privados).

- **Contenedores**: Son instancias ejecutables y aisladas de las imágenes. Usan tecnologías del kernel Linux como **namespaces** (para aislamiento de procesos, red, usuarios, etc.) y **cgroups** (control de recursos como CPU, memoria y I/O).

APIs (Interfaz de Programación de Aplicaciones)

Una API (Application Programming Interface) es un conjunto de definiciones, protocolos y herramientas para construir y conectar software. Técnicamente, es un contrato que especifica cómo deben interactuar los componentes de software mediante llamadas, solicitudes y respuestas.

- **Tipos**:
    - **APIs RESTful**: Utilizan HTTP para comunicación, empleando métodos como GET, POST, PUT, DELETE. Se basan en recursos identificados por URIs y utilizan formatos estándar (JSON, XML).
    - **APIs SOAP**: Protocolo basado en XML y mensajes estructurados para aplicaciones distribuidas.
    - **APIs GraphQL**: Permiten consultas flexibles y obtención de solo los datos necesarios.
- **Componentes técnicos**:
    - **Endpoints**: URLs que representan recursos o servicios específicos.
    - **Métodos HTTP**: Definen la acción a realizar.

- **Headers**: Para pasar metadata como autenticación (tokens JWT, OAuth), tipo de contenido, etc.

- **Códigos de estado HTTP**: Informan sobre el resultado de la petición (200 OK, 401 Unauthorized, 404 Not Found, 500 Internal Server Error).

IoT (Internet of Things)

El **Internet de las Cosas (IoT)** es un paradigma tecnológico donde dispositivos físicos (“cosas”) con sensores, actuadores, software y conectividad de red están interconectados para recopilar, transmitir y actuar sobre datos en tiempo real.

- **Arquitectura técnica**:

    - **Dispositivos finales (Edge devices)**: Sensores y actuadores con microcontroladores o microprocesadores embebidos que pueden tener sistemas operativos ligeros (FreeRTOS, Zephyr).

    - **Conectividad**: Protocolos específicos y optimizados para bajo consumo y ancho de banda limitado (MQTT, CoAP, LwM2M, ZigBee, LoRaWAN).

    - **Gateways**: Puentes que agrupan y filtran datos, realizan procesamiento local (edge computing) y comunican con la nube.

    - **Plataformas IoT en la nube**: Proveen almacenamiento masivo, análisis de datos, dashboards y APIs para gestión remota (AWS IoT, Azure IoT Hub, Google IoT Core).

Node-RED

Node-RED es una herramienta de programación visual basada en flujo, orientada a la integración de sistemas y el desarrollo rápido de aplicaciones IoT y middleware.

Fundamentos técnicos:

- Construido sobre **Node.js**, aprovechando su modelo de eventos asíncrono y no bloqueante.

- Proporciona un **editor basado en navegador** para diseñar flujos mediante la conexión de nodos que representan entradas, salidas,

transformaciones y servicios.

Arquitectura de Node-RED:

- Los flujos se componen de nodos que pasan mensajes JSON.
- Cada nodo puede ejecutar código JavaScript personalizado, conectarse a APIs externas, bases de datos, o dispositivos físicos.
- Los mensajes viajan a través del flujo en tiempo real, permitiendo procesamiento, filtrado, enriquecimiento y orquestación.

Casos de uso técnicos:

- Integración de dispositivos IoT y protocolos heterogéneos.
- Orquestación de servicios y automatización industrial.
- Rapid prototyping de soluciones IoT sin necesidad de escribir código desde cero.

Desarrollo

A continuación se describe el procedimiento técnico llevado a cabo para la instalación del motor de contenedores Docker en un sistema Ubuntu.

Esta etapa forma parte fundamental del Proyecto Final Capstone, cuyo objetivo es integrar soluciones de IoT mediante flujos visuales en un entorno modular, portable y replicable.

2. Justificación del uso de Docker:

Docker es una herramienta de virtualización ligera que permite empaquetar aplicaciones y sus dependencias en contenedores. Al contenerizar Node-RED, se garantiza: - Portabilidad entre sistemas. - Aislamiento del entorno de ejecución. - Facilidad de despliegue y mantenimiento. - Escalabilidad del sistema IoT.

3. Requisitos previos:

- Sistema operativo: Ubuntu.

- Acceso con privilegios de superusuario.

- Conexión a Internet.

- Terminal con privilegios sudo.

4. Instalación de Docker en Ubuntu:

(pasos ejecutados)

1. Actualización del sistema:

```
sudo apt update && sudo apt upgrade -y
```

2. Instalación de dependencias necesarias:

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

3. Añadir la clave GPG oficial de Docker:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor
-o /usr/share/keyrings/docker.gpg
```

4. Añadir el repositorio de Docker:

```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null
```

5. Instalación del motor Docker:

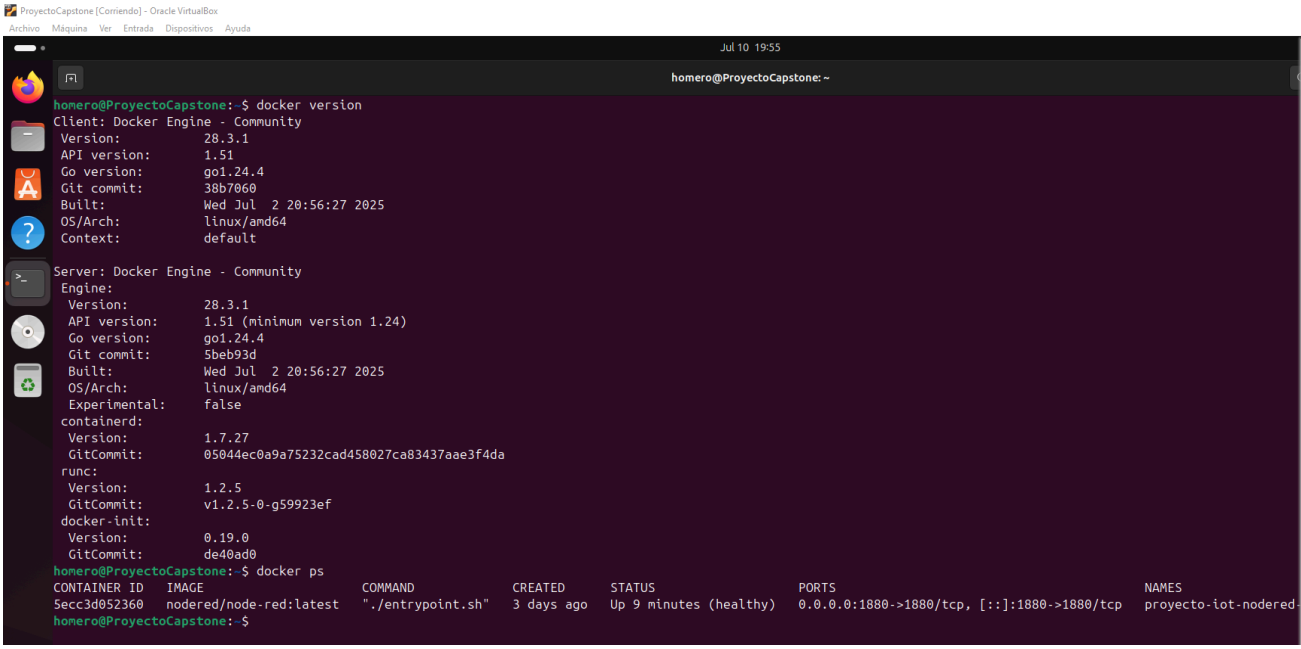
```
sudo apt update sudo apt install docker-ce docker-ce-cli containerd.io -y
```

6. Verificación del servicio Docker:

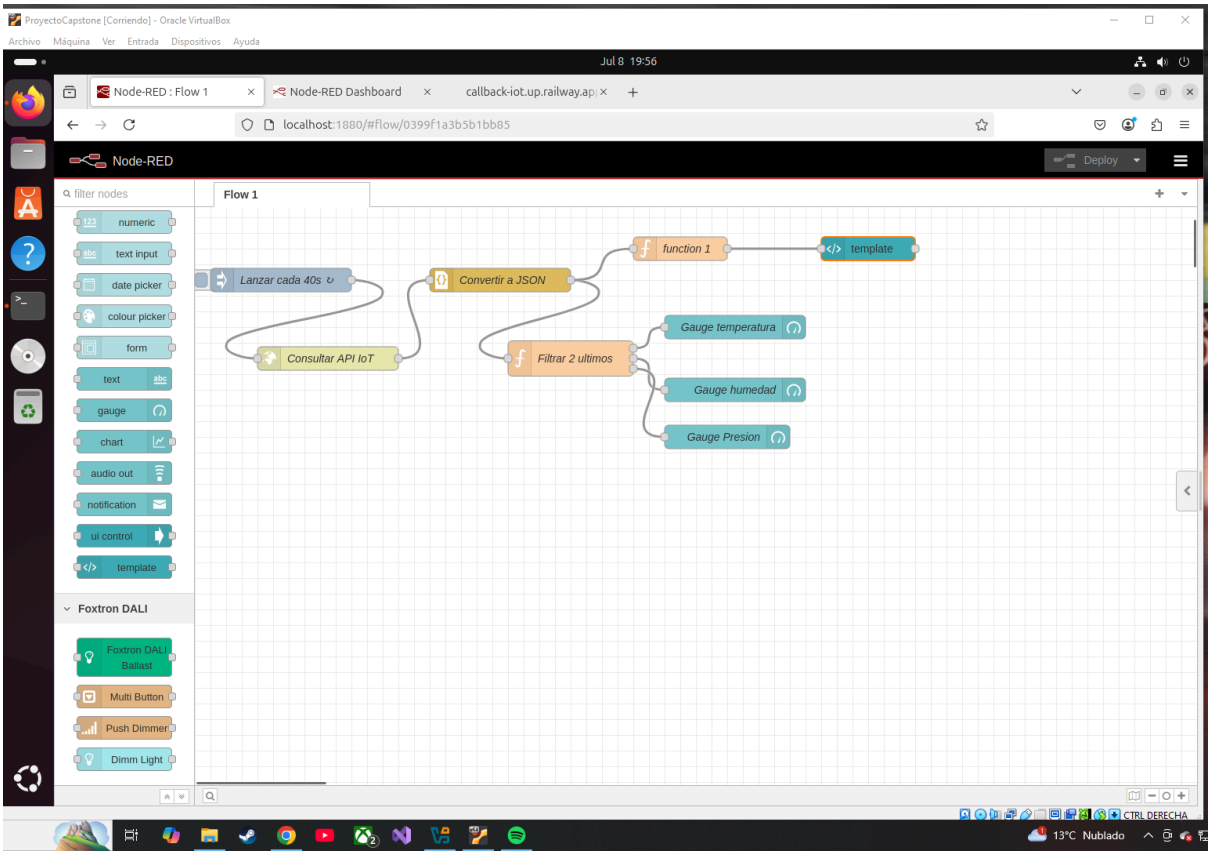
```
sudo systemctl status docker
```

7. Agregar el usuario actual al grupo docker :

```
sudo usermod -aG docker $USER
```



Flujo de Node-RED



El flujo construido en Node-RED permite consumir datos provenientes de una API REST, procesarlos, visualizarlos en tiempo real mediante indicadores (gauges) y mostrar un resumen de las últimas mediciones en una tabla dinámica. El flujo se compone de seis nodos principales organizados de manera secuencial y funcional.

1. Nodo Inject

El nodo *inject* es el punto de partida del flujo. Su función principal es disparar manual o automáticamente la ejecución del mismo. Puede configurarse para activarse de forma periódica (por ejemplo, cada minuto) o mediante un clic manual del usuario. Este nodo genera una señal de activación que inicia el proceso de adquisición de datos desde la API.

2. Nodo HTTP Request

Una vez activado el flujo por el nodo inject, el nodo *http request* se encarga de realizar una solicitud a un endpoint expuesto por una API REST. Este endpoint entrega datos provenientes de un sistema IoT, típicamente en formato JSON, que contienen registros de variables ambientales como temperatura, humedad y presión atmosférica. La solicitud es de tipo GET y retorna una respuesta que será utilizada en los siguientes pasos del flujo.

3. Nodo JSON

El nodo *json* tiene la función de convertir la respuesta recibida desde la API (que generalmente es un texto en formato JSON) a un objeto de datos estructurado que pueda ser manipulado por los nodos siguientes. Este paso es fundamental, ya que permite que el contenido de la respuesta sea tratado como un conjunto de datos dinámicos, sobre los cuales se pueden aplicar operaciones lógicas o de filtrado.

4. Nodo Function para los Gauges

A continuación, un primer nodo *function* toma el conjunto de datos estructurados y extrae la última medición disponible, que corresponde al valor más reciente registrado por el dispositivo IoT. Esta medición contiene las tres variables de interés: temperatura, humedad y presión. El nodo procesa estos valores y los distribuye a tres indicadores visuales tipo gauge, uno por cada variable. Estos gauges están

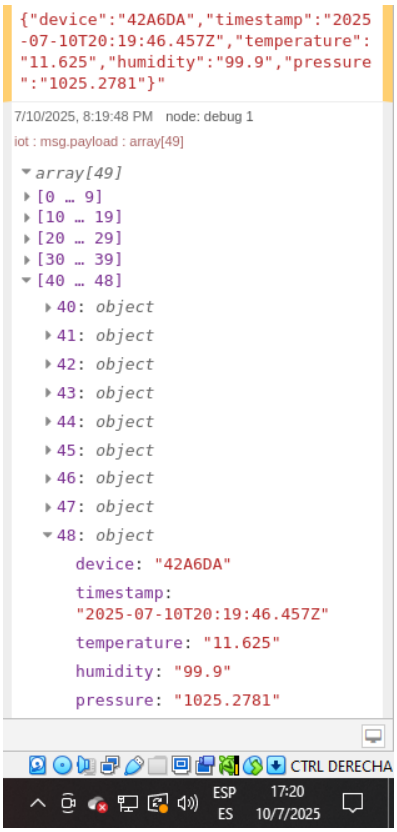
integrados en el dashboard de Node-RED y permiten al usuario visualizar en tiempo real el estado actual de las condiciones medidas.

5. Nodo Function para mostrar los dos últimos registros

El segundo nodo *function* cumple la función de generar un resumen más amplio de las mediciones recientes. En este caso, el nodo selecciona las dos últimas entradas del conjunto de datos. El objetivo es construir una tabla comparativa que permita observar los cambios recientes en las variables ambientales. Estos datos se preparan para ser utilizados por un nodo de presentación.

6. Nodo Template para HTML

Finalmente, un nodo *template* toma los datos procesados por el segundo nodo function y los utiliza para generar contenido HTML dinámico. Este contenido se visualiza en el dashboard en forma de una tabla que presenta los dos registros más recientes de temperatura, humedad y presión. Este componente del flujo está orientado a la interfaz de usuario y mejora la presentación y comprensión de los datos, facilitando la comparación y el análisis visual de las condiciones registradas.



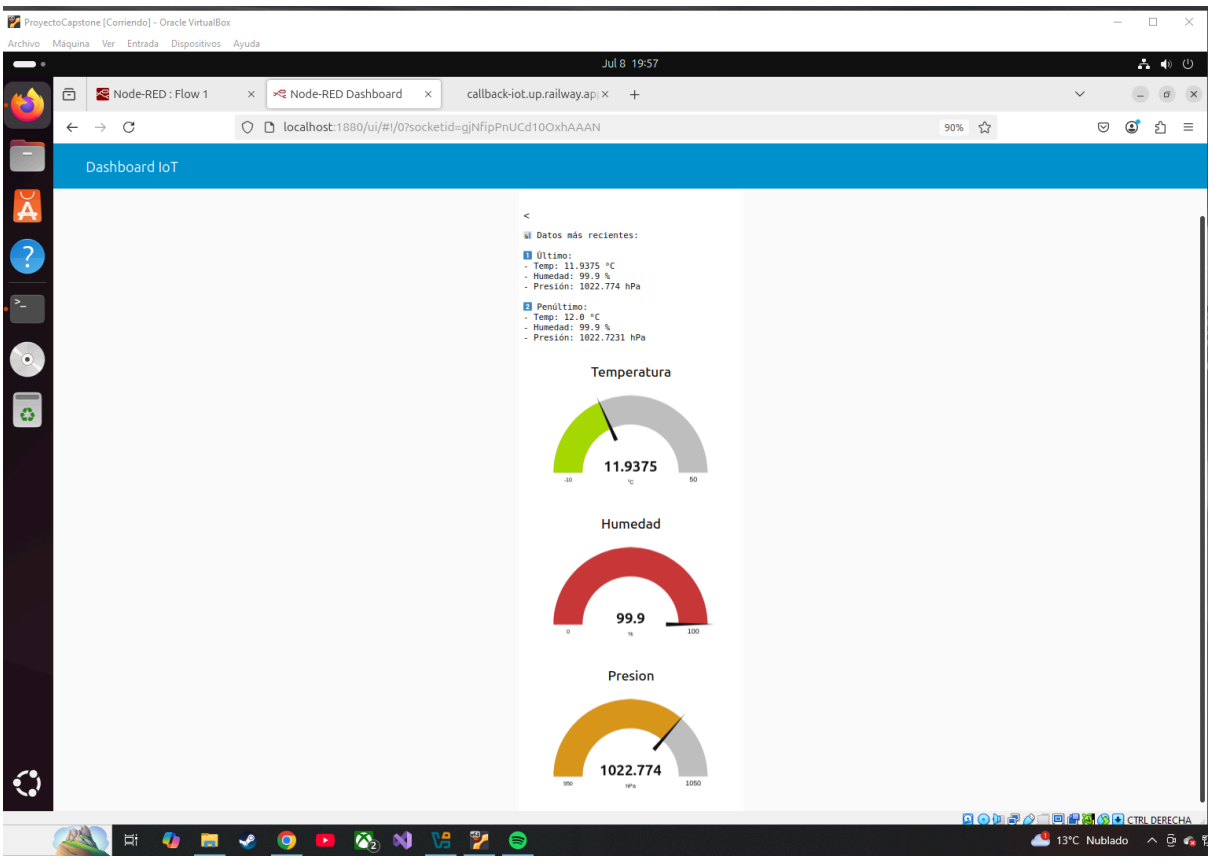
Ejemplo del JSON recibido

Conclusión del funcionamiento general

El flujo completo permite, de forma automatizada o manual, obtener datos desde una fuente externa (API), transformarlos y procesarlos para ser representados en un entorno visual interactivo. Utilizando herramientas de bajo código como Node-RED, este sistema cumple con los requisitos funcionales de una solución IoT básica de monitoreo, integrando adquisición, procesamiento y visualización de datos en tiempo real dentro de un entorno contenerizado (Docker).

Conclusión final

El flujo desarrollado en Node-RED permite automatizar la adquisición de datos desde una API externa, procesarlos en tiempo real y visualizarlos mediante una interfaz web interactiva. Este funcionamiento puede observarse en la captura del dashboard, que representa fielmente los datos recogidos y procesados por el sistema.



El resultado es un entorno visual accesible, que refleja de manera fiel el procesamiento del flujo en tiempo real. La integración entre el backend de Node-RED y la interfaz del dashboard demuestra el correcto funcionamiento del sistema, validando su capacidad para adquirir, transformar y mostrar datos de forma autónoma y eficiente.

Reflexión final del proyecto

Llevar adelante este proyecto fue una experiencia intensa, desafiante y formativa. Desde el inicio, enfrenté la dificultad de integrar múltiples tecnologías con las que, en algunos casos, tenía poca o ninguna experiencia previa. Node-RED, Docker, el manejo de APIs REST y conceptos de IoT eran temas que conocía más en teoría que en la práctica.

Uno de los principales desafíos fue entender cómo funcionaba Docker a nivel técnico. Al principio me resultaba abstracto el concepto de "contenedor", pero con el tiempo fui entendiendo su importancia para crear entornos aislados y reproducibles. Instalar Docker correctamente, manejar imágenes y volúmenes, y contenerizar Node-RED fueron pasos que me obligaron a investigar a fondo, cometer errores y aprender de ellos.

También me costó estructurar el flujo en Node-RED. Aunque su entorno visual facilita el desarrollo, traducir la lógica que tenía en mente a nodos funcionales requería tiempo, pruebas y depuración. Una de las mayores frustraciones fue lograr que los datos obtenidos desde la API REST se visualizaran correctamente en el dashboard, especialmente cuando debía separar los valores y mostrarlos de manera simultánea en gauges y tablas.

Sin embargo, todas esas dificultades se transformaron en aprendizajes. Ahora entiendo con mucha más claridad cómo fluye la información en un sistema IoT real, desde el sensor o simulador hasta su visualización en un dashboard. Aprendí a trabajar con datos en formato JSON, a desplegar servicios en contenedores y a resolver problemas técnicos de forma autónoma.

Este proyecto me dejó más que un resultado funcional: me dio herramientas concretas para el futuro, tanto para mi formación profesional como para mi desarrollo personal, y me asoma más hacia el entorno virtual en el que se desarrolla el rubro de la carrera que estoy estudiando, cada día toma más relevancia en mi mente lo que es Linux, los entornos de desarrollo, y cómo trabajar en base al inmenso mundo que es la programación.