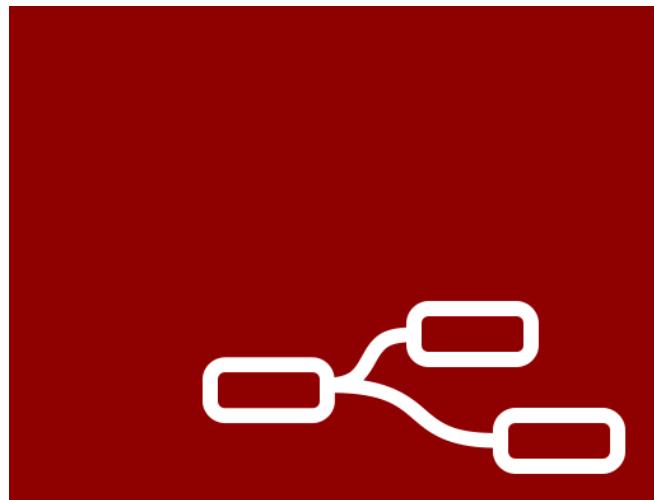


Sistemas Industriales

Practica 1



Node-RED

Elvi Mihai Sabau Sabau^{1[51254875L]}

¹ Universidad de Alicante, Alicante, España.
emss5@alu.ua.es

Abstract. En este documento se redacta todas las tareas realizadas durante las sesiones de teoría para el primer bloque entregable, además de otros apartados extra expuestos por cada sesión.

1 Sesión 1.	3
1.1 Introducción de la sesión	3
1.2 Instalacion de node-red	3
1.3 Ejercicio.	7
1.3.1 API Key.	7
1.3.2 Consumiendo la API con node-red.	9
1.3.3 Añadiendo una función JS.	11
1.3.4 Instalado la Palettes de OpenWeatherMap y usando sus nodos.	12
2 Sesión 2	15
2.2 Generación de variables con números aleatorios.	15
2.4 Apartado Extra: Solicitud del tiempo.	20

3 Sesión 3	23
3.1 Sobre los reles.	23
3.2 La practica.	24
3.3 En mas detalle.	25
4 Sesión 4	26
4.1 Nodos a usar.	26
4.2 Emitiendo y capturando datos.	27
4.3 Integración con shelly.	27
4.4 Apartado extra: despliegue de un broker MQTT usando Docker y Mosquitto.	
4.4.1 Configuración	28
4.4.2 Arranque	28
4.5 Trabajo Paralelo: Especificación técnica para el proyecto de enfermería.	29
5 Sesión 5.	29
6 Sesión 6.	29
6.1 Interacción con el sensor Xiaomi.	30
6.1.1 Detección del sensor.	30
6.1.2 Configuración del script.	30
6.1.4 Conexión desde Node-red.	31
6.2 Ubidots.	32
6.2.1 Ubidots usando un dashboard y un script en python.	32
6.2.2 Ubidots usando node-red.	35

1 Sesión 1.

1.1 Introducción de la sesión

En esta primera sesión nos hemos familiarizado con el entorno de trabajo de bajo código llamado node-red.

1.2 Instalacion de node-red

Para arrancar este entorno tenemos 2 opciones:

1.2.1 Mediante npm.

Podemos descargar el servidor de node-red desde los repositorios de npm, para que este se instale directamente en nuestro sistema.

Usando el siguiente comando, podremos instalar node-red y ejecutar el comando “node-red” para arrancar el entorno.

```
sudo npm install -g --unsafe-perm node-red
```

Podemos después confirmar que se ha instalado correctamente si vemos este mensaje.

```
+ node-red@1.1.0
added 332 packages from 341 contributors in 18.494s
found 0 vulnerabilities
```

Ahora, podremos ejecutar node-red desde la terminal ejecutando el comando “node-red”.

```
node-red
20 Sep 10:15:09 - [info] Node-RED version: v3.0.2
20 Sep 10:15:09 - [info] Node.js  version: v16.17.0
20 Sep 10:15:09 - [info] Linux 5.15.0-46-generic x64 LE
20 Sep 10:15:10 - [info] Loading palette nodes
20 Sep 10:15:11 - [info] Settings file : /home/frenzoid/.node-red/settings.js
20 Sep 10:15:11 - [info] Context store : 'default' [module=memory]
20 Sep 10:15:11 - [info] User directory : /home/frenzoid/.node-red
20 Sep 10:15:11 - [warn] Projects disabled : editorTheme.projects.enabled=false
20 Sep 10:15:11 - [info] Flows file   : /home/frenzoid/.node-red/flows.json
20 Sep 10:15:11 - [info] Server now running at http://127.0.0.1:1880/
20 Sep 10:15:11 - [warn]

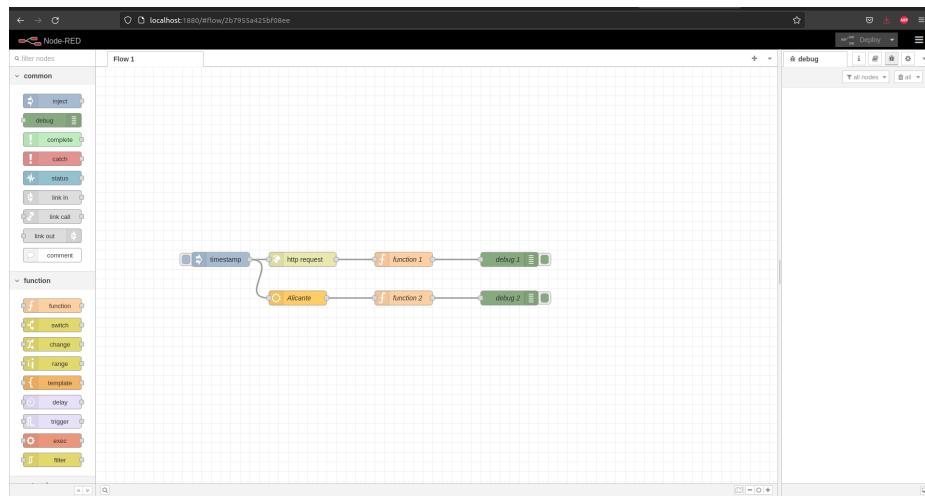
-----
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.

-----
20 Sep 10:15:11 - [info] Starting flows
20 Sep 10:15:11 - [info] Started flows
```

Y ahora, desde el navegador, si navegamos a `localhost:1880` podremos acceder a nuestro entorno de node-red.



1.2.2 Mediante docker.

Podemos arrancar un simple contenedor docker con el siguiente comando, según la documentación:

```
docker run -it -p 1880:1880 --name mynodered nodered/node-red
```

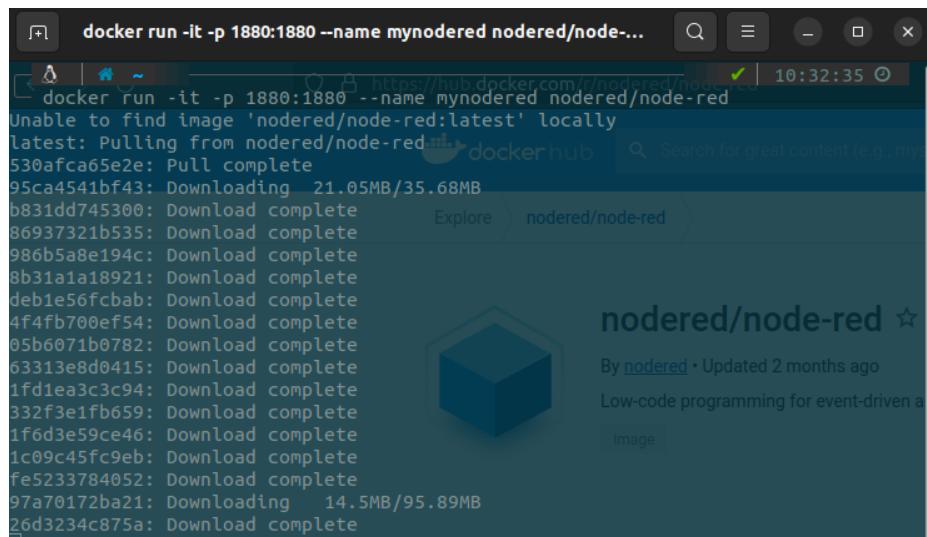
Lo que hace este comando es arrancar un contenedor con la imagen de nodered/node-red de manera interactiva y mapeando el puerto 1880 del contenedor al puerto 1880 del ordenador anfitrión.

El problema que aporta este comando en concreto es que el contenedor no tiene persistencia, al apagar el contendor se perderán todos los cambios, así que para que tenga persistencia vamos a crear un volumen llamado “node_red_dara”, y ejecutar el siguiente comando.

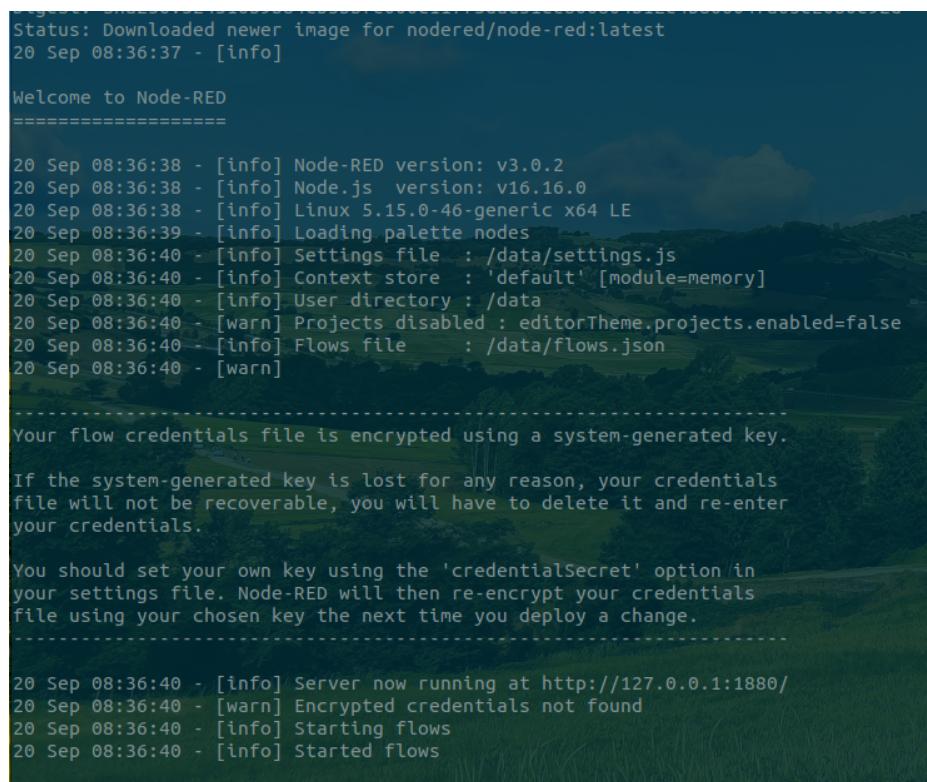
```
docker run -it -p 1880:1880 -v node_red_data:/data --name mynodered nodered/node-red
```

De esta manera, nuestro contenedor tendrá persistencia de datos.

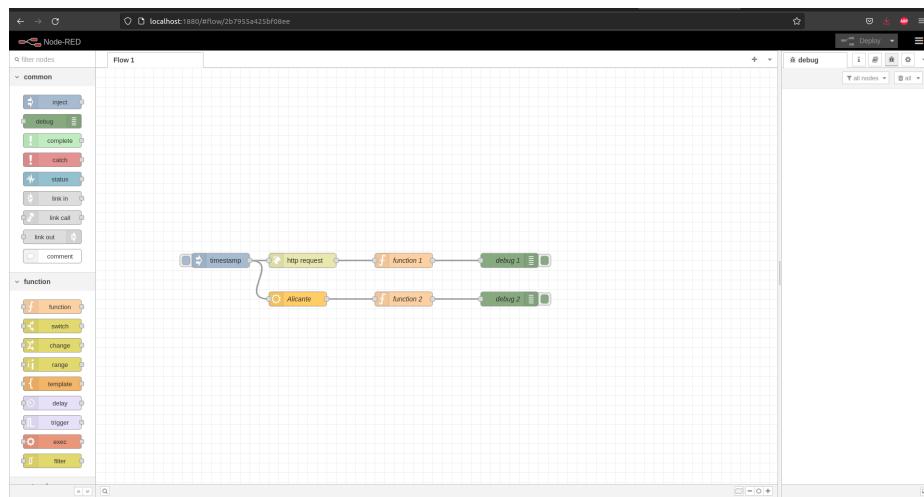
Ejecutamos el comando, y veremos que al no tener la imagen de node-red descargada, se nos empezará a descargar automáticamente.



Una vez acabada la descarga de la imagen, empezará a arrancar el contenedor.



Y ahora, desde el navegador, si navegamos a `localhost:1880` podremos acceder a nuestro entorno de node-red.



1.3 Ejercicio.

En esta sesión el ejercicio que hemos realizado es el de consumir una API mediante nodos de node-red. La API a consumir es la de OpenWeather. Consumiremos esta API de dos formas, mediante un nodo que nos permita realizar peticiones HTTP y mediante un nodo que nos permite interactuar directamente con OpenWeather.

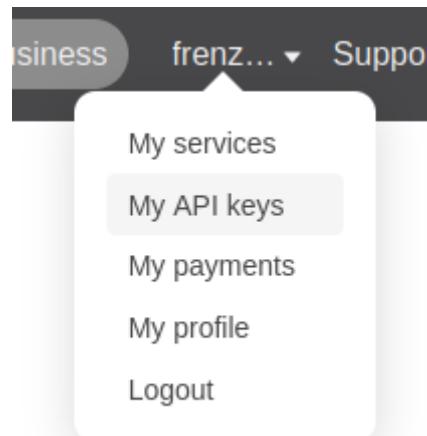
1.3.1 API Key.

Primero, para esto, nos registramos en OpenWeather, y obtendremos una API Key. Una API Key es una llave que se acopla a la url de la petición, y nos permite consumir el servidor APIRest de forma limitada, la API Key se usa para evitar abusos sobre el uso de la API.

Nos registramos e iniciamos sesión.

<input type="text"/>	frenzoid
<input type="password"/>	*****
<input checked="" type="checkbox"/> Remember me	
<input type="button" value="Submit"/>	

Ahora accedemos a “My API Keys”.



Y ahora, en esta pestanya podemos generar nuestras API Keys.

Key	Name	Status	Actions	Create key
b2aa1c7d35ba53c6b3d5df387b2f2f3d	sdfsdf	Active	<input type="button" value="Edit"/> <input type="button" value="Delete"/>	<input type="text" value="API key name"/> <input type="button" value="Generate"/>

Pero como las usamos? Buscando en la documentación podemos encontrar esta pagina que nos explica el uso de la API de Open Weather.

<https://openweathermap.org/current>

Call current weather data

How to make an API call

API call

`https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}`

Parameters

`lat, lon` required Geographical coordinates (latitude, longitude). If you need the geocoder to automatically convert city names and zip-codes to geo coordinates and the other way around, please use our [Geocoding API](#).

`appid` required Your unique API key (you can always find it on your account page under the "API key" tab)

`mode` optional Response format. Possible values are `xml` and `html`. If you don't use the `mode` parameter format is JSON by default. [Learn more](#)

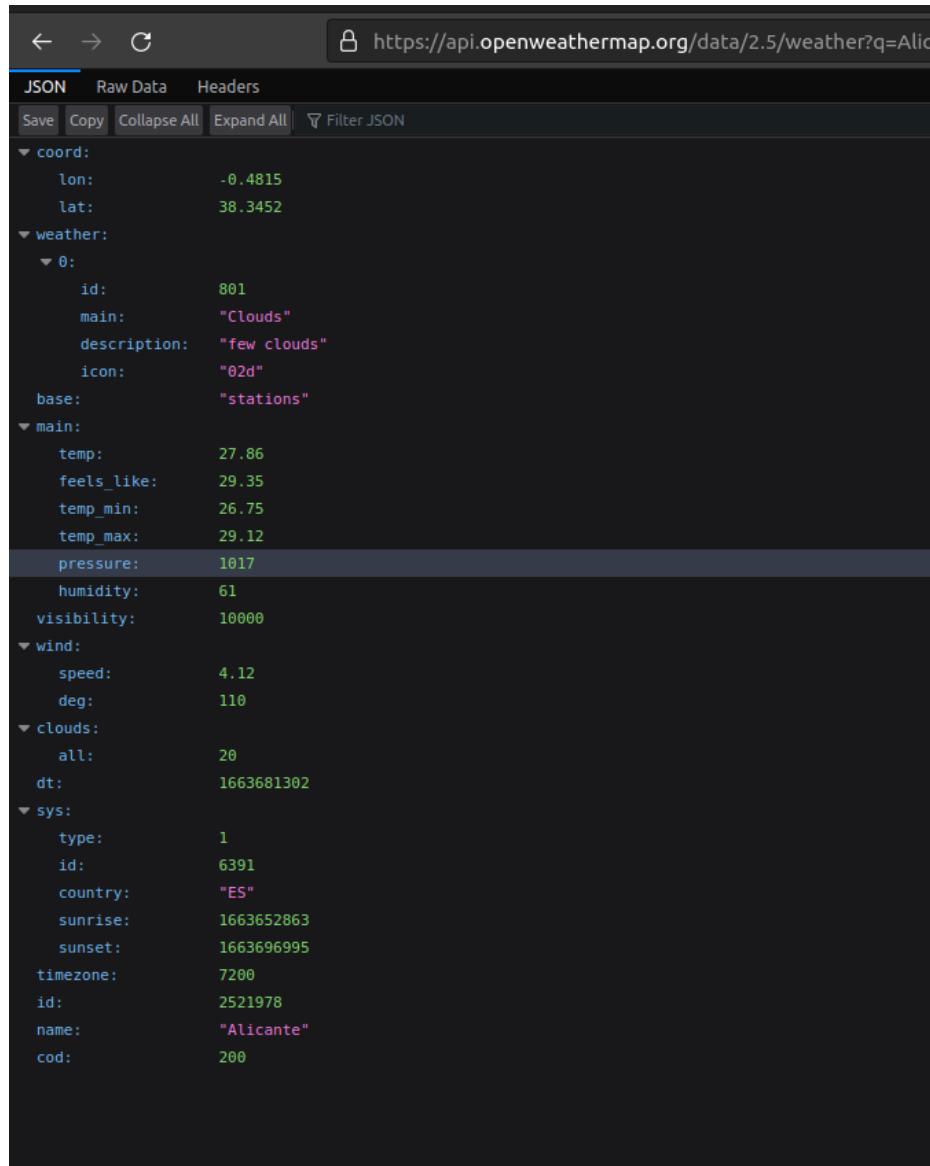
`units` optional Units of measurement. `standard`, `metric` and `imperial` units are available. If you do not use the `units` parameter, `standard` units will be applied by default. [Learn more](#)

`lang` optional You can use this parameter to get the output in your language. [Learn more](#)

You can use the geocoder built into this API by default, but please note that this version is less accurate than the [Geocoder API](#) and will be deprecated soon. [Learn more](#)

Sabiendo esto, nosotros podemos probar a obtener la situación meteorológica de Alicante en unidades métricas con simplemente accediendo a esta url.

<https://api.openweathermap.org/data/2.5/weather?q=Alicante&appid=b2aa1c7d35ba53c6b3d5df387b2f2f3d&units=metric>



The screenshot shows a JSON viewer interface with the URL <https://api.openweathermap.org/data/2.5/weather?q=Alicante&appid=b2aa1c7d35ba53c6b3d5df387b2f2f3d&units=metric>. The JSON data is displayed in a hierarchical tree structure:

```

{
  "coord": {
    "lon": -0.4815,
    "lat": 38.3452
  },
  "weather": [
    {
      "id": 801,
      "main": "Clouds",
      "description": "few clouds",
      "icon": "02d"
    }
  ],
  "main": {
    "temp": 27.86,
    "feels_like": 29.35,
    "temp_min": 26.75,
    "temp_max": 29.12,
    "pressure": 1017,
    "humidity": 61,
    "visibility": 10000
  },
  "wind": {
    "speed": 4.12,
    "deg": 110
  },
  "clouds": {
    "all": 20
  },
  "dt": 1663681302,
  "sys": {
    "type": 1,
    "id": 6391,
    "country": "ES",
    "sunrise": 1663652863,
    "sunset": 1663696995,
    "timezone": 7200,
    "id": 2521978,
    "name": "Alicante",
    "cod": 200
  }
}

```

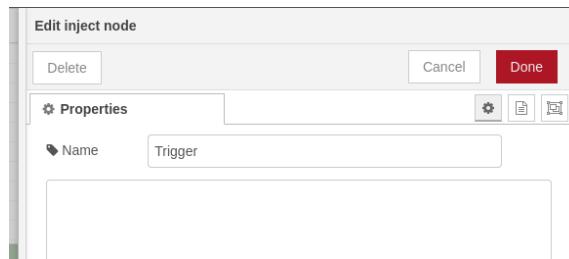
Ahora veremos cómo procesar estos datos mediante node-red.

1.3.2 Consumiendo la API con node-red.

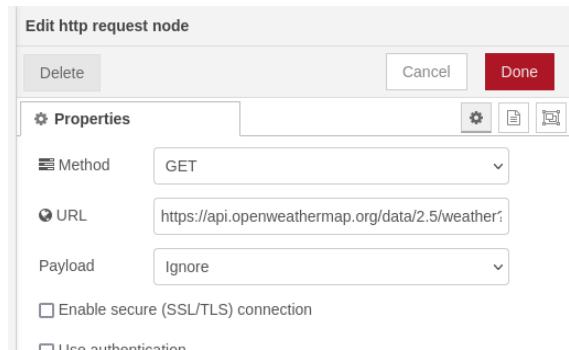
Para esto, vamos a usar 3 nodos, un nodo “Inject”, otro nodo “http request” y por último un nodo “debug”.



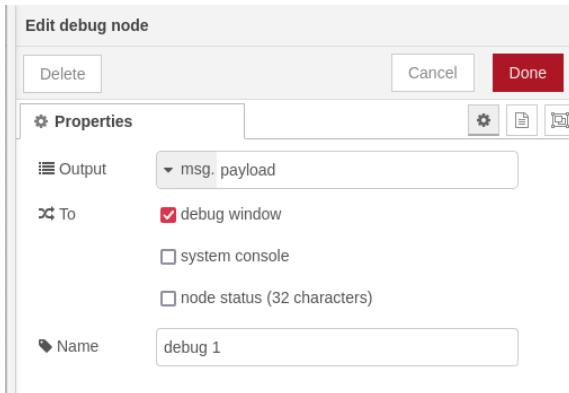
El nodo inject por defecto enviará al siguiente nodo un dato, en concreto la fecha actual, este valor se puede cambiar, pero a nosotros nos interesa usarlo como botón para ejecutar el siguiente nodo. En mi he limpiado el nodo, y lo uso simplemente de botón.



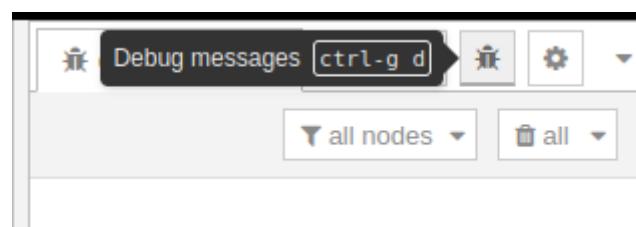
El nodo http request nos permite realizar peticiones http y mandar la respuesta al siguiente nodo. En la configuración de este nodo especificaremos el método GET y la URL que mencionamos en el apartado anterior.



El nodo debug nos permite mostrar los datos recibidos por la pantalla de depuración, podemos mostrar todo el objeto, o sólo una parte, siendo esta por defecto “payload”.



Para acceder a la pantalla debug debemos apretar en este botón en la columna derecha.



Una vez hecho este esquema con la configuración mencionada anteriormente, al ejecutar el injector, podremos ver los datos solicitados a la API en la pantalla de depuración.

```

9/20/2022, 4:02:27 PM node: debug 1
msg.payload : Object
  ▼ object
    ▶ coord: object
    ▶ weather: array[1]
      base: "stations"
    ▶ main: object
      visibility: 10000
    ▶ wind: object
    ▶ clouds: object
    dt: 1663682408
    ▶ sys: object
      timezone: 7200
      id: 2521978
      name: "Alicante"
      cod: 200
  
```

Vamos ahora a ver cómo podemos procesar este objeto para mostrar solo la temperatura.

1.3.3 Añadiendo una función JS.

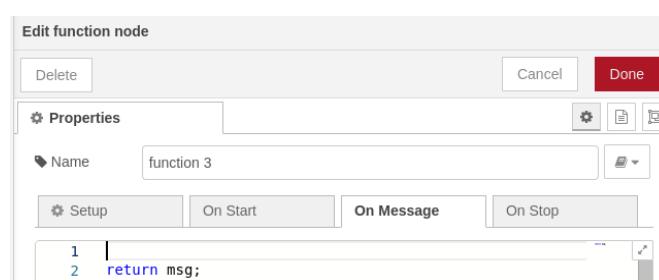
Mediante el nodo “function” nosotros recibiremos los datos en una variable llamada “msg” que contiene un objeto llamado “payload” que contiene el valor emitido por el nodo anterior.



Si encargamos el nodo “http request” a este nodo, y la salida en el debug, podremos interceptar el dato y modificarlo antes de mostrarlo.

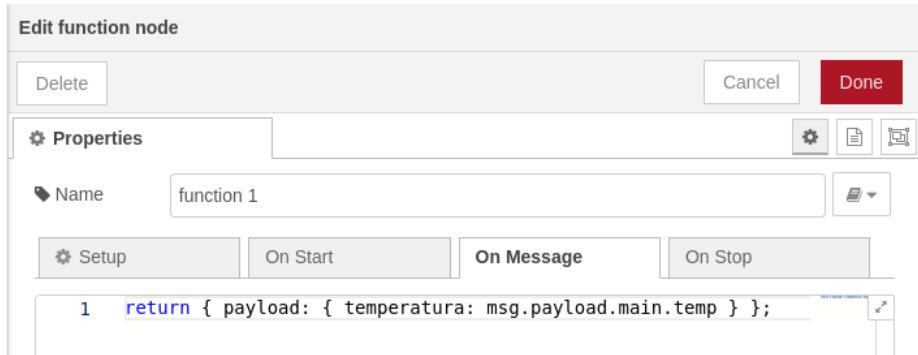


El valor devuelto por esta función será el valor que se le pasará al siguiente nodo enganchado a este nodo.

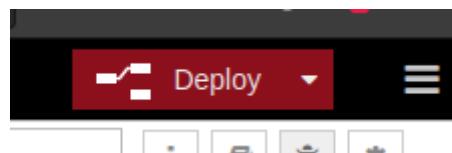


Este nodo soporta también ejecutar lógica antes y después del despliegue del propio nodo.

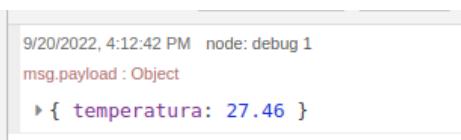
Ahora en este nodo, vamos a filtrar el dato a devolver, siendo el valor devuelto un objeto con la clave “temperatura” y el valor siendo la temperatura de la ciudad.



Y ahora, podemos desplegar los nodos desde el botón “Deploy”.



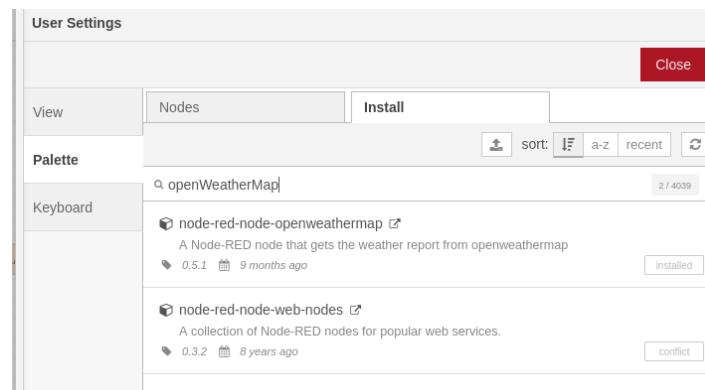
Una vez desplegados los nodos, podremos interactuar con el trigger, y obtendremos estos datos en la pantalla de depuración.



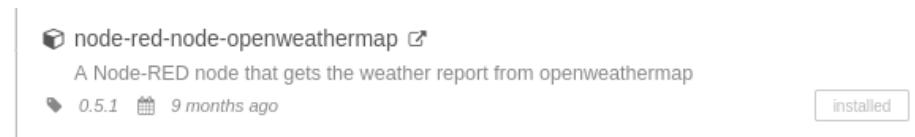
Ahora veremos cómo consumir la API usando un nodo de OpenWeatherMap.

1.3.4 Instalado la Palettes de OpenWeatherMap y usando sus nodos.

Node-red permite usar otros nodos aparte de los nodos que vienen por defecto, estos nodos se pueden descargar como librerías desde el gestor de palettes, para abrir el gestor de palettes se puede hacer o desde el panel lateral > palette, o apretando los botones “alt” + “shift” + “p”.



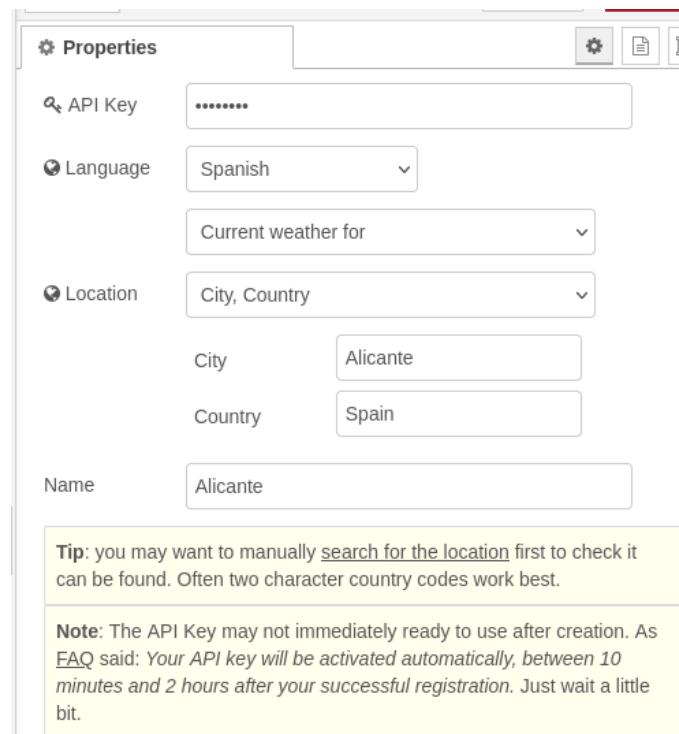
El paquete que vamos a instalar es el siguiente:



Una vez instalado, tendremos esta sección con estos nodos disponibles:

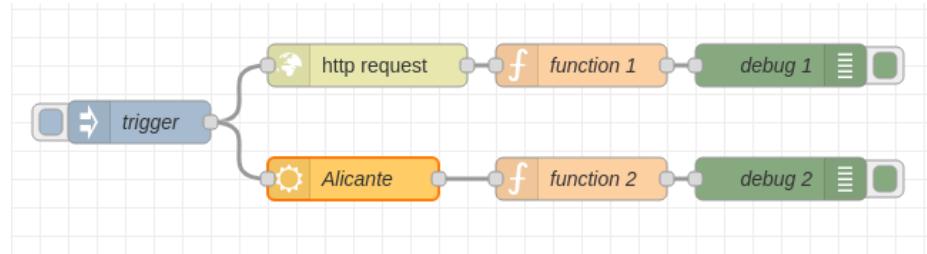


Estos nodos nos permiten configurar nuestra API Key, el lenguaje de la respuesta, y la ciudad para la que queremos obtener los datos. La única diferencia es que la primera se ejecuta de manera manual, y la segunda se ejecuta automáticamente cuando un dato específico cambia.



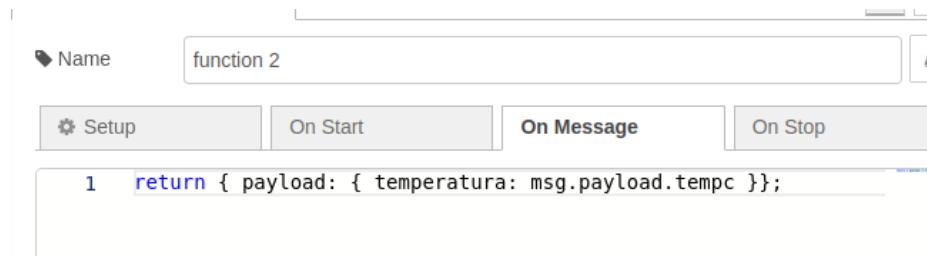
Nosotros usaremos el primer nodo, y lo configuraremos como se muestra en la foto anterior.

Una vez hecho esto lo enganchamos y creamos otros nodos, de tal manera que se quedaría similar al flujo anterior.



En este segundo caso, el dato devuelto tiene un formato diferente que cuando consumimos la api usando el nodo “http request”.

Para ello, vamos a escribir un código diferente en “función 2”, para obtener el mismo resultado en la pantalla de depuración.



Ahora, vamos a desplegar, y a ejecutar el inyector, veremos ambos resultados en la pantalla de depuración.



2 Sesion 2

2.1 Dependencias.

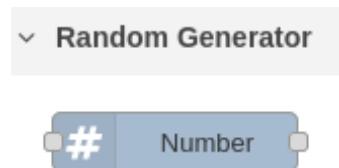
Para esta sesión, vamos a instalar las siguientes dependencias.

node-red	3.0.2	> 49 nodes	in use
node-red-contrib-ui-led	0.4.11	> 1 node	remove disable all
node-red-dashboard	3.2.0	> 21 nodes	in use
node-red-node-openweathermap	0.5.1	> 2 nodes	in use
random-generator_node-red-contrib	1.0.0	> 6 nodes	in use

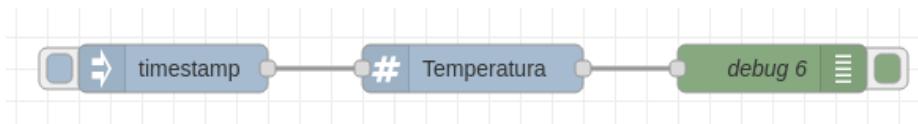
2.2 Generación de variables con números aleatorios.

En esta sesión hemos implementado un mecanismo para generar números aleatorios, y asignarlos a unas variables, para poder posteriormente usarlos en otros mecanismos de nuestro flujo.

Para ello, vamos a usar el nodo Random Generator > Number.



Y creamos un flujo sencillo para probarlo. El número aleatorio en este caso representará una temperatura.



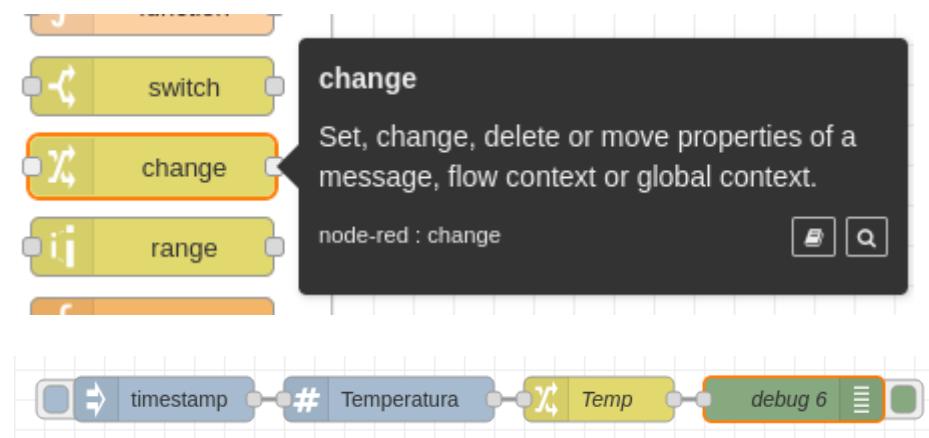
Podemos configurar el valor del rango del número generado.

Name	Temperatura
Minimum	10
Maximum	30
Number of Decimals	2
Only Integers	<input checked="" type="checkbox"/>

Ahora, si ejecutamos este flujo, veremos los resultados.

```
9/27/2022, 9:17:34 AM node: debug 6
msg.payload : number
22
9/27/2022, 9:17:35 AM node: debug 6
msg.payload : number
26
9/27/2022, 9:17:35 AM node: debug 6
msg.payload : number
30
```

Ahora, para crear una variable y asignarle el valor aleatorio usaremos este nodo.



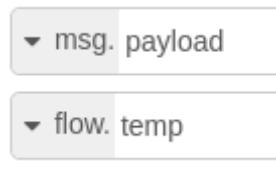
Y para crear una variable, en la configuración de este nodo tendremos 2 campos, 1) destino, 2) origen. El destino es el nombre de la variable a la que se le asignará el valor del origen.

En este caso para crear una variable llamada “temp” haremos lo siguiente:

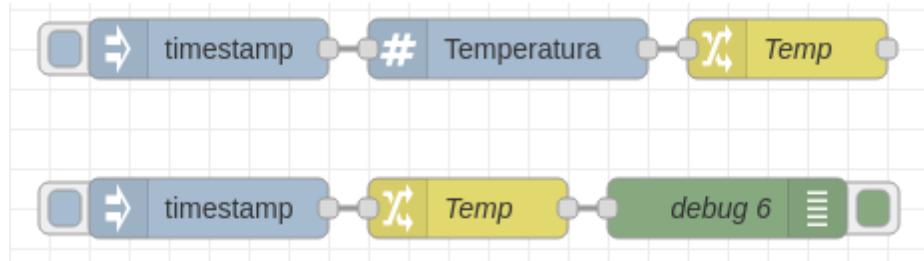


Ahora, para probarlo, vamos a crear 2 flujos, uno que asigne, y otro que lea de la variable.

Para leer, solo tenemos que cambiar de sentido de donde leemos y a donde asignamos:



`msg.payload = flow.temp`



Así, podemos ver el resultado.

```

9/27/2022, 9:25:37 AM node: debug 6
msg.payload : number
15
9/27/2022, 9:25:39 AM node: debug 6
msg.payload : number
13
  
```

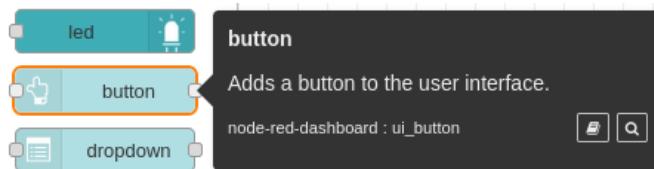
2.3 Despliegue de una interfaz web.

En este apartado vamos a crear una pequeña interfaz web, donde vamos a poder interactuar con la generación de números aleatorios.

Para esto vamos a usar los siguiente nodos:



Este nodo “gauge” nos permitirá tener un widget en nuestra página web similar a un medidor, es totalmente configurable, tanto el color como la temática de este

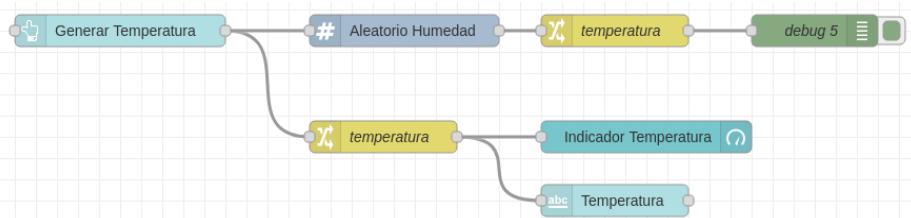


Este nodo nos permitirá tener un botón en nuestra página web, con la cual interactuar en nuestro flujo.



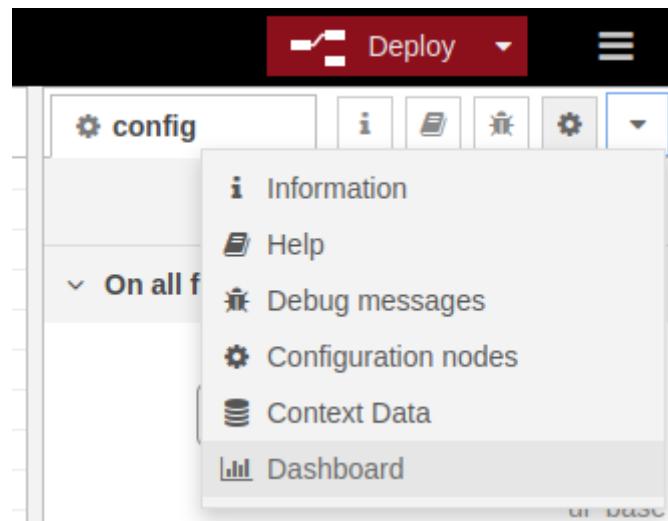
Este nodo es una simple caja de texto, que nos permite imprimir un texto en nuestra página web.

y creamos el siguiente flujo:

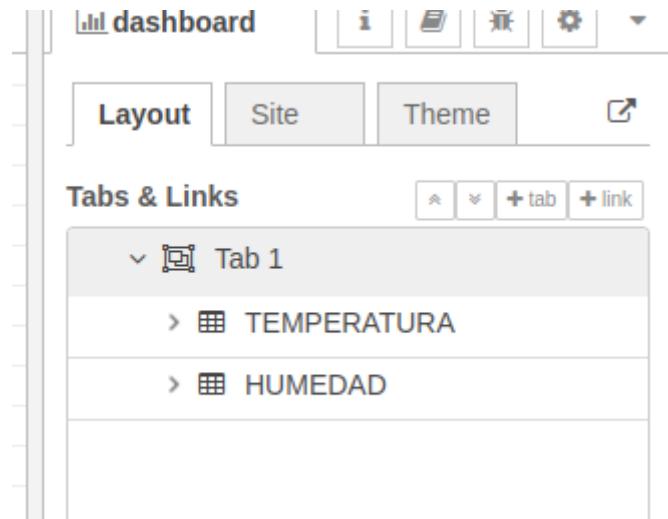


Aun así, si accedemos a 127.0.0.1:1880/ui no veremos nada, esto se debe a que la página web se gestiona mediante pestañas y submenús, y estos al no estar asociados a ninguno no se van a representar en la interfaz.

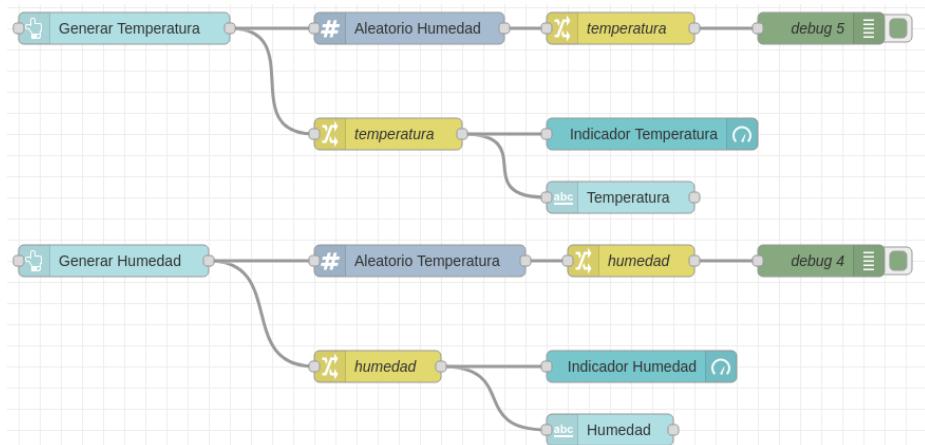
Para ello, primero vamos a crear una pestaña. Esto lo haremos desde: subpestañas > Dashboards > + tab.



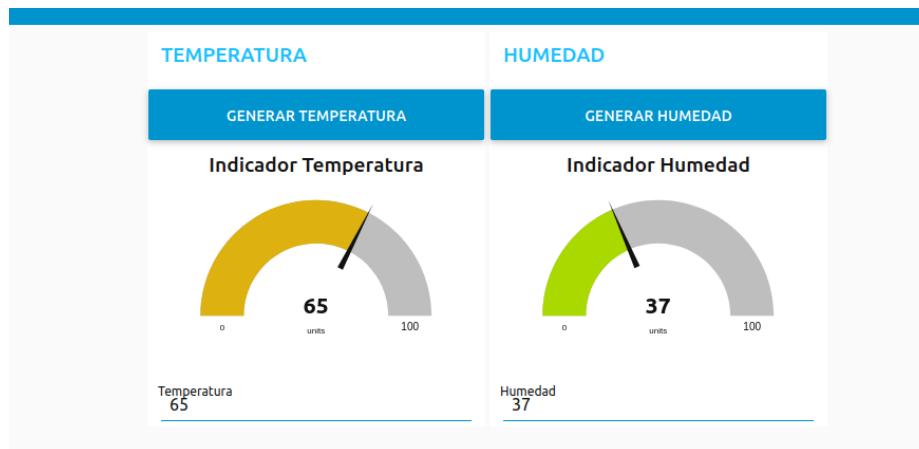
En mi caso, yo tengo estas pestañas:



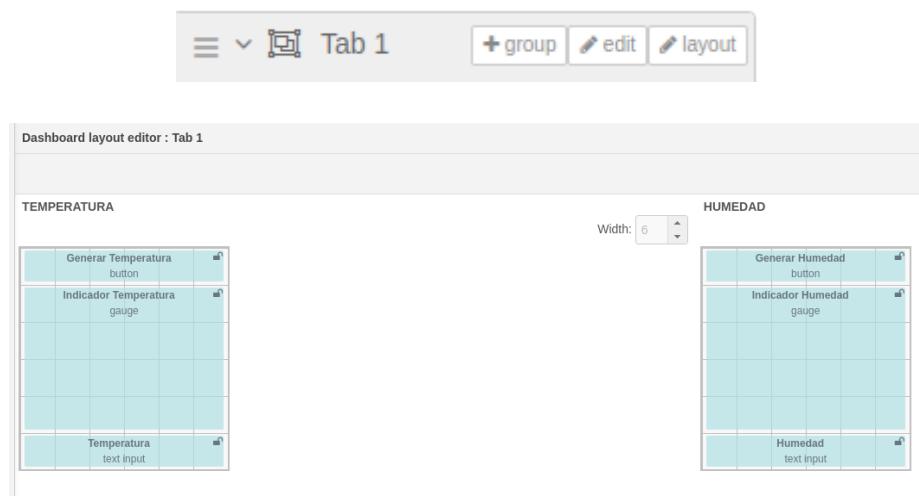
Y tengo cada nodo widget del par de flujos siguiente asociado a una pestaña diferente.



De esta manera, si configuramos cada widget a una pestaña, ya podremos ver los widgets asociados de cada nodo.

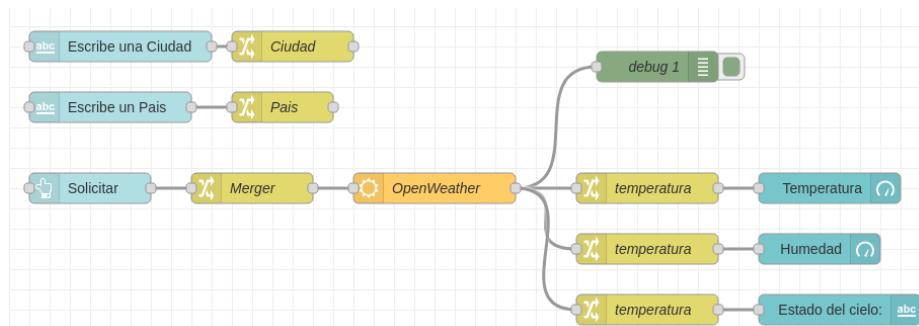


Además, desde Layout, podemos ordenar en qué forma se muestran los widgets.

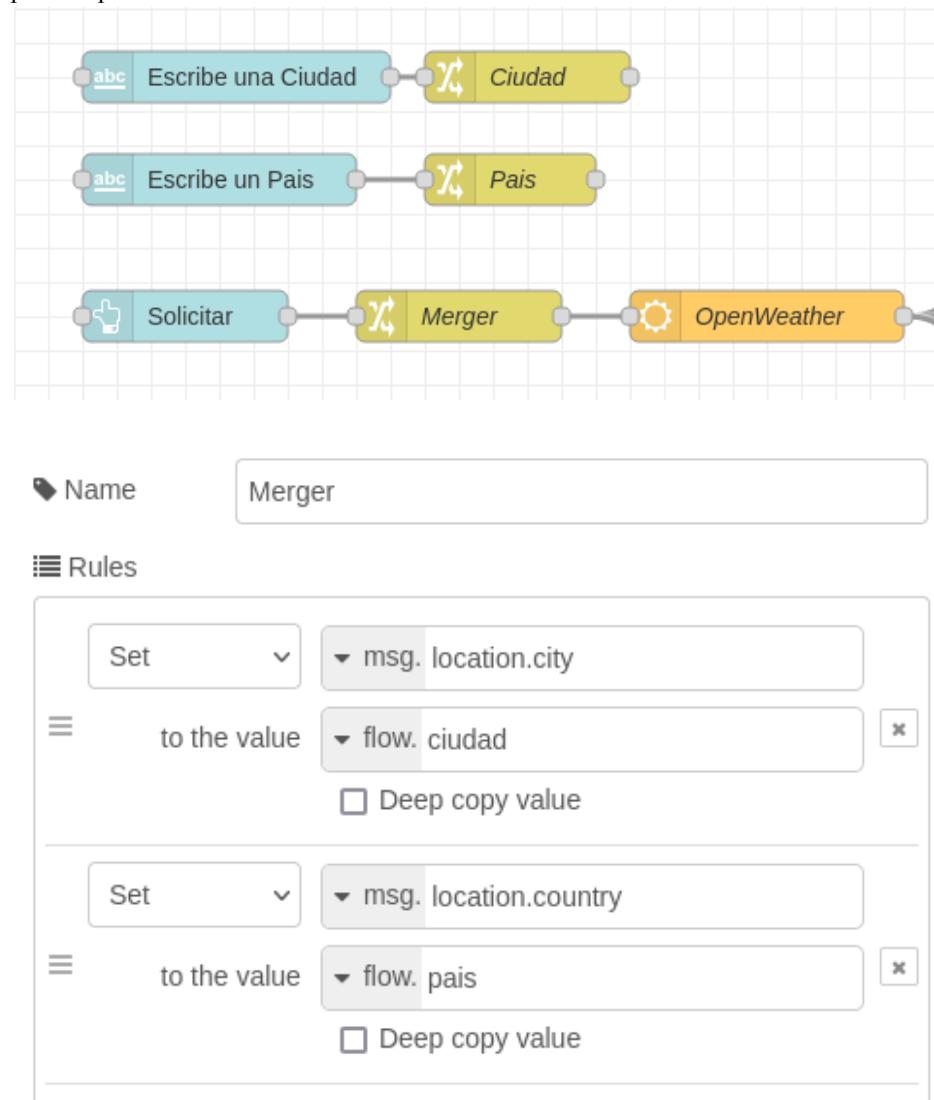


2.4 Apartado Extra: Solicitud del tiempo.

Como apartado extra, he creado un pequeño formulario donde podemos introducir la ciudad y el país, y obtendremos la temperatura, humedad y estado del cielo.



Este es el flujo que he creado, en merger junto las variables ciudad y país, y se las paso a open weather.



Mi dashboard será similar al ejemplo mencionado en el punto 2.2.

Tabs & Links

- Tab 1
 - SELECCION
 - Escribe un País
 - Escribe una Ciudad
 - Solicitar
 - ESTADO
 - Estado del cielo:
 - Temperatura
 - Humedad

Y este es el resultado de mi dashboard:

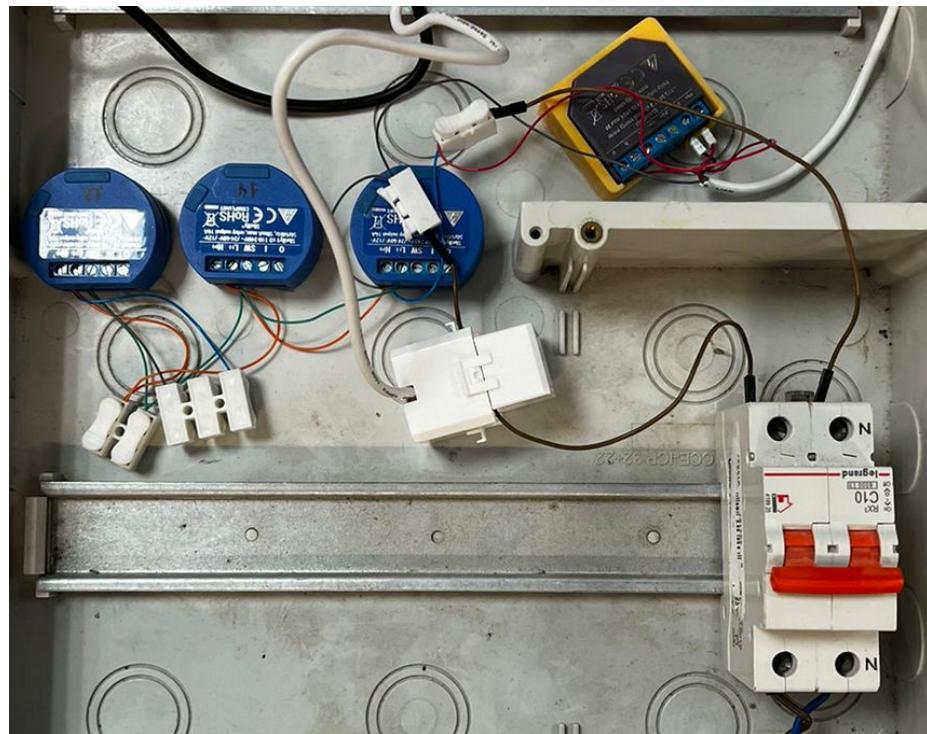


3 Sesión 3

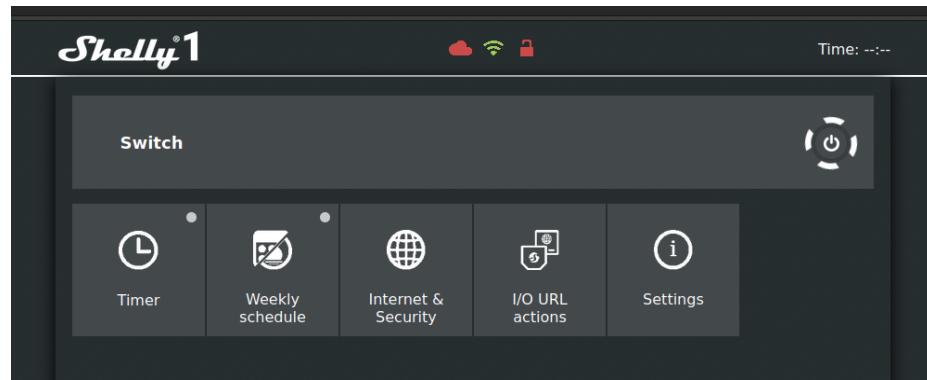
En esta sesión hemos aprendido a interactuar con unos relés que pueden conectarse a la red y vía HTTP interactuar con estos.

3.1 Sobre los reles.

Estos relés de la marca Shelly son relés inteligentes. Estos permiten mediante internet, ser manipulados remotamente, además de solicitar el estado de estos.

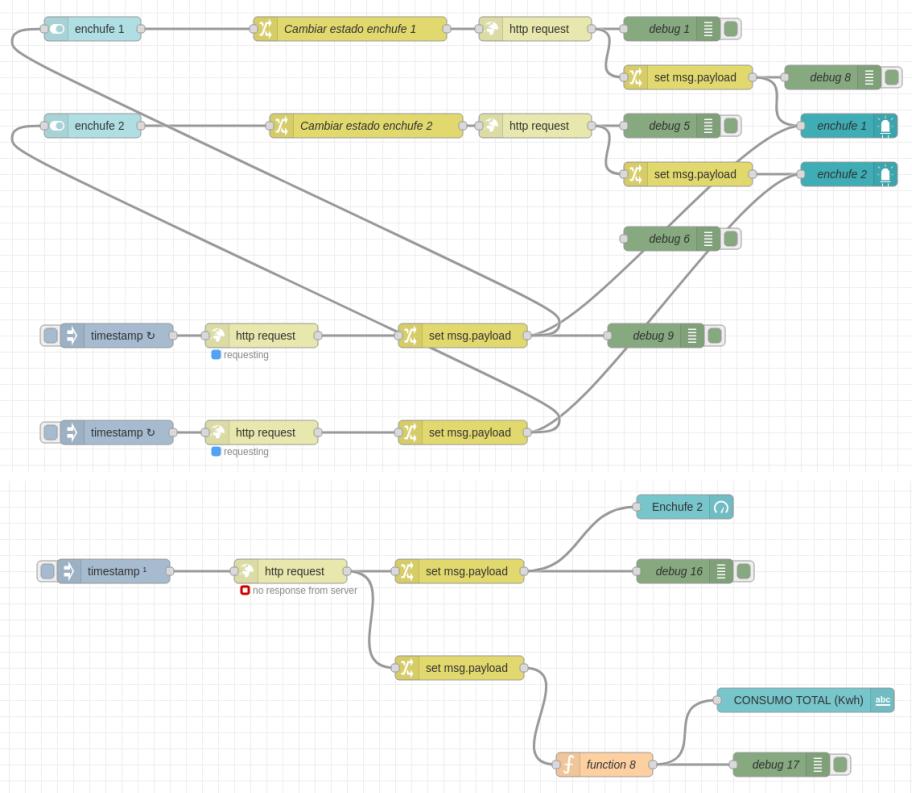


Estas poseen una API Rest mediante la cual podemos solicitar estos datos en formato JSON, o interactuar con el relé, además poseen una interfaz gráfica desde la cual podemos interactuar con el relé directamente.



3.2 La practica.

La práctica fue bastante interesante, mediante node-red hemos logrado interactuar con la API de estos relés, además de tener una interfaz propia que nos permite interactuar con varios relés.



Estos flujos en node-red nos permiten renderizar una interfaz capaz de encender / apagar y ver el consumo actual de unos enchufes.

(La siguiente foto de la interfaz está deshabilitada por el hecho de que la he tomado cuando en casa, mientras hacía la memoria, sin tener acceso a los relés)



3.3 En mas detalle.

Estos reles sirven para poder automatizar factores previamente analogicos, que requerían una interacción manual.

Esto último lo hemos visto en detalle al visitar el invernadero de la Universidad de Alicante, ubicado en el parque científico. En este sitio hemos visto la automatización mediante una raspberry, relés inteligentes y otros dispositivos la automatización del mantenimiento y cultivo de las plantas allí ubicadas.



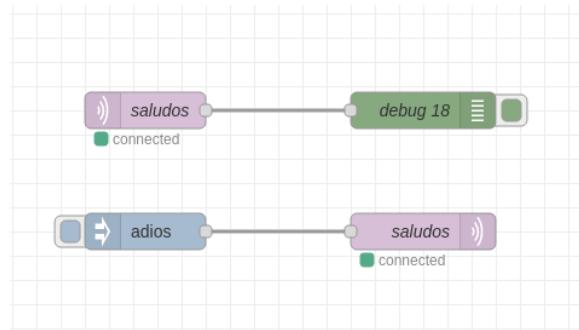
Estos relés tienen mucho potencial, pero una mala configuración pueden volverlos en un gran punto débil de seguridad, por ello se recomienda encarecidamente que la red a la cual se conectan estos relés sea una red propia sin acceso al exterior, en concreto en una DMZ o intranet, y usar un intermediario como node-red para interactuar desde el exterior.

4 Sesion 4

En esta sesión hemos aprendido a cómo conectarnos a un broker MQTT mediante node red, y cómo interactuar con los reles Shelly a través de MQTT. Además como parte extra, he arrancado un broker MQTT en docker, ya que la raspberry del profesor no funcionaba.

4.1 Nodos a usar.

Los nodos que usaremos serán 2, un productor y un consumidor de mensajes MQTT. El emisor lo enganchamos a un nodo trigger, y el receptor a un nodo debugger, para poder ver los datos.



Configuraremos los nodos MQTT para que usen un nuevo broker, y lo enganchamos a un tópico, en este caso “saludo”.



The screenshot shows a configuration interface for an MQTT client. At the top, there's a 'Name' field set to 'LOCAL'. Below it, a tab bar has 'Connection' selected, with other tabs for 'Security' and 'Messages'. Under 'Connection', the 'Server' field contains '172.27.70.64', and the 'Port' field is set to '1883'. There are two checkboxes: one checked for 'Connect automatically' and one unchecked for 'Use TLS'. The 'Protocol' dropdown is set to 'MQTT V3.1.1'. The 'Client ID' field is set to 'Leave blank for auto generated'. The 'Keep Alive' field is set to '60'. Finally, under 'Session', there's a checked checkbox for 'Use clean session'.

El broker usado es un broker que he desplegado localmente en un contenedor docker, y que no requiere credenciales para acceder.

4.2 Emitiendo y capturando datos.

Podemos configurar el mensaje a enviar desde el trigger, este pasará un mensaje a emitir por el productor.



Y lo capturamos con el consumidor, mostrándolo en la consola usando el nodo debug.

```
10/11/2022, 9:20:05 AM node: debug 18
saludos : msg.payload : string[5]
"adios"
```

4.3 Integración con shelly.

Hemos visto además, que los reeles inteligentes pueden usar un bróker MQTT como interfaz entre otros dispositivos. La nomenclatura de cada dispositivo suele ser un tópico que sigue la sintaxis de “<modelo>-<mac>”, aunque este puede ser configurable, el tópico después se parte en subtópicos dependiendo del tipo de dato que se quiera solicitar (al suscribirnos a dicho tópico), o emitir para configurar el relee. También podemos suscribirnos a todos los subtópicos o tópicos usando la almohadilla.

4.4 Apartado extra: despliegue de un broker MQTT usando Docker y Mosquitto.

Desplegar un broker MQTT mediante docker es bastante fácil.

4.4.1 Configuración

Primero crearemos un archivo llamado “mosquitto.conf”, dentro de este archivo añadiremos 3 líneas:

- listener 1883
- persistence true
- allow_anonymous true

listener: especifica en que puerto escuchar.

persistence: especifica la persistencia de los mensajes recibidos.

allow_anonymous: especifica el acceso anónimo (sin credenciales).

4.4.2 Arranque

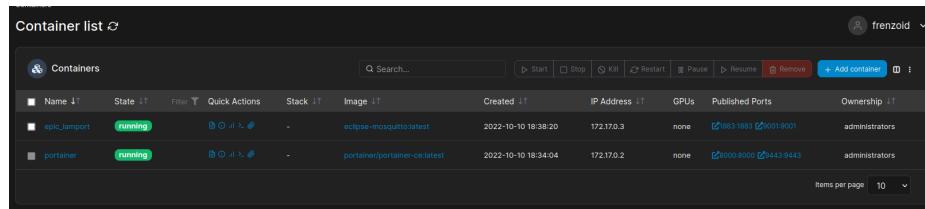
Para arrancar, solo tenemos que ejecutar el siguiente comando:

```
docker run -it -p 1883:1883 -p 9001:9001 -v
mosquitto.conf:/mosquitto/config/mosquitto.conf
eclipse-mosquitto
```

Sobre los parametros:

- -it: permite la ejecución interactiva.
- -p: redirige un puerto del ordenador a uno del contenedor.
- -v: monta un archivo del ordenador en el contenedor.

Una vez hecho esto, ya tendremos nuestro broker en línea, y la ip a usar será la del ordenador.



The screenshot shows a "Log viewer settings" interface with the following configuration:

- Auto-refresh logs: On
- Wrap lines: On
- Display timestamps: Off
- Fetch: All logs
- Search: Filter...
- Lines: 100
- Actions: Download logs, Copy, Copy selected lines, Unselect

The log entries listed are:

```

1665420488: New connection from 192.168.5.114:62556 on port 1883.
1665420488: New client connected from 192.168.5.114:62556 as nodered_73674edb4c90977e (p2, c1, k60).
1665420488: Client nodered_b9ad408e6fb7642 disconnected.
1665420488: New connection from 192.168.5.103:62662 on port 1883.
1665420488: New client connected from 192.168.5.103:62662 as nodered_21541aed0bce30cc (p2, c1, k60).
1665420489: New connection from 192.168.5.111:54386 on port 1883.
1665420489: New client connected from 192.168.5.111:54386 as nodered_c98937ccccf86616d (p2, c1, k60).
1665420489: Client nodered_54ebc9e2bb077ac8 disconnected.
1665420497: New connection from 192.168.5.101:50657 on port 1883.
1665420497: New client connected from 192.168.5.101:50657 as nodered_3fd6d9aa0f93cd704 (p2, c1, k60).
1665420500: New connection from 192.168.5.103:62673 on port 1883.
1665420500: New client connected from 192.168.5.103:62673 as nodered_48130aa22604a35e (p2, c1, k60).
1665420501: Client nodered_21541aed0bce30cc disconnected.
1665420501: New connection from 192.168.5.115:40996 on port 1883.
1665420501: New client connected from 192.168.5.115:40996 as nodered_a2a5a410c4ccb3c8 (p2, c1, k60).
1665420501: Client nodered_11ea6656051b75bf disconnected.
1665420506: New connection from 192.168.5.105:60614 on port 1883.
1665420506: Client nodered_dd8776c308f36fb disconnected.
1665420509: Client nodered_11b4aefcd9283313 disconnected due to malformed packet.
1665420509: New connection from 192.168.5.107:63829 on port 1883.
1665420509: New client connected from 192.168.5.107:63829 as nodered_f22f6eb67eb8b715 (p2, c1, k60).
1665420510: Client nodered_e6ff7da5cf2ca25 disconnected.
1665420518: New connection from 192.168.5.109:57614 on port 1883.
1665420518: New client connected from 192.168.5.109:57614 as nodered_b8314dee6b7af27a (p2, c1, k60).

```

4.5 Trabajo Paralelo: Especificación técnica para el proyecto de enfermería.

Esta especificación se puede encontrar al final de este documento.

5 Sesión 5.

En la sesión 5 hemos intentado realizar una práctica con un sensor Xiaomi que emite sus datos vía BLE. Esta práctica no se pudo realizar debido a que no sabíamos la MAC del sensor, por ello, se continuó en la sesión 6.

6 Sesión 6.

En esta sesión se particiona en 2 prácticas:

- Leemos de un sensor Xiaomi de humedad y temperatura vía BLE a través de un script en Python. Después conectamos este script a un nodo node-red “exec” y enviamos los datos vía MQTT a un broker que desplegué en mi ordenador. Después con un consumidor MQTT mostraremos los datos en el apartado de debug de noder-red.
- Nos crearemos una cuenta en [Ubidots](#), nos crearemos un nuevo dashboard, y mediante un código en python mandaremos datos de muestra a nuestro

dashboard de ubidots, para mostrar dichos datos de una forma fácilmente interpretable. Además aprenderemos a usar el nodo ubidots de node-red para mandar datos a la dashboard desde node-red.

6.1 Interacción con el sensor Xiaomi.

6.1.1 Detección del sensor.

El sensor en cuestión es un sensor de Xiaomi, capaz de medir la temperatura y la humedad.



Para detectar la señal que este sensor manda cada cierto tiempo, hemos usando la herramienta “hcitool” para detectar señales de BLE (Bluetooth Low Energy).

```
sudo hcitool lescan
LE Scan ...
2C:BA:99:95:AE:51 (unknown)
78:72:C4:A4:B1:24 (unknown)
35:AC:94:40:2F:19 (unknown)
C0:09:0A:02:4F:0E (unknown)
C0:09:0A:02:4F:0E Mi Smart Band 5
34:5A:6B:75:EE:76 (unknown)
74:C4:08:64:B3:8D (unknown)
47:93:5E:A3:E0:54 (unknown)
4F:D5:A2:67:E3:7A (unknown)
6D:C4:9E:AA:81:35 (unknown)
```

6.1.2 Configuración del script.

Una vez obtenida la mac, procedemos a clonar el siguiente [repositorio](#), este repositorio provee un script en python que lee los datos del sensor, y los emite los emite a un servidor MQTT.

Una vez clonado, para cambiar la configuración por defecto cambiaremos los contenidos de los archivos “device.ini” y “mqtt.ini” (una vez clonado, estos archivos tendrán un prefijo “.sample”, borraremos dicho prefijo del nombre de cada archivo).

Primero vamos a editar “device.ini” especificando la mac y el tópico al cual queremos emitir los mensajes.

```
⚙️ devices.ini
1 [xiaomiSensor]
2 device_mac = 58:2D:34:31:44:07
3 topic = sensor/laboratorio
4 availability_topic = sensor/laboratorio/availability
5 average = 3
6 timeout = 5
```

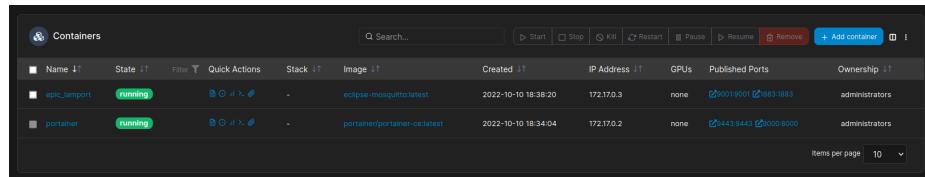
Y después editaremos “mqtt.ini” especificando la dirección del broker MQTT, el puerto y las credenciales.

```
⚙️ mqtt.ini
1 [broker]
2 client=xiaomi-ble
3 host=localhost
4 port=1883
5 username=
6 password=
```

Una vez hecho esto, ya podremos leer los datos del sensor, ahora vamos a arrancar el broker MQTT.

6.1.3 Arranque del broker MQTT.

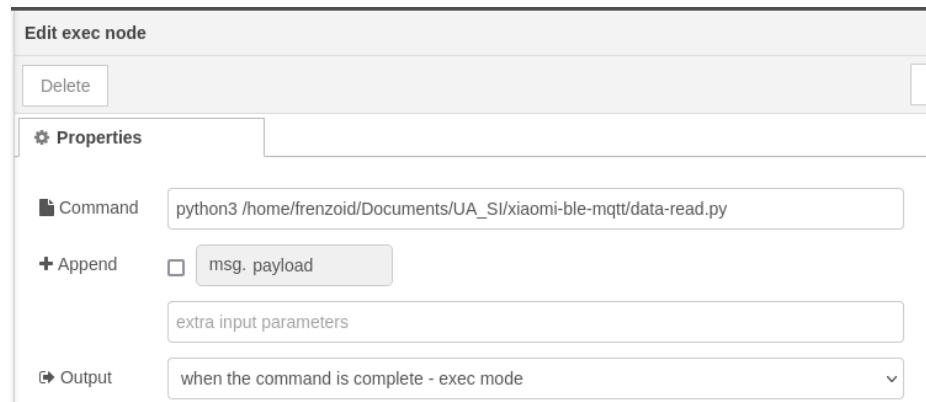
Como este broker ya lo tenemos creado de la práctica anterior, solo tendremos que encender el contenedor.



Una vez arrancado, ahora vamos a nuestro panel de node-red.

6.1.4 Conexión desde Node-red.

Aquí, lo que haremos será enlazar un nodo trigger a un exec, y en este exec, especificaremos el comando y el archivo a ejecutar, en este caso:



Nuestro flow de nodos será el siguiente.



De esta manera, podremos ejecutar el script en python desde node-red, y como este script emite el dato del sensor vía MQTT a nuestro broker, lo que haremos será crear otro flow donde capturaremos este mensaje desde el tópico especificado en "devices.ini".

```
sensor/laboratorio : msg.payload : Object
  ↪ { temperature: 26.5, humidity: 78.2, battery: 69, average: 3 }
```

6.2 Ubidots.

6.2.1 Ubidots usando un dashboard y un script en python.

Primero, nos crearemos una cuenta trial en ubidots, una vez registrados nos crearemos un dashboard desde: menu > dashboards > +.

The top screenshot shows the 'Add new Dashboard' configuration page. It has two tabs: 'SETTINGS' (selected) and 'APPEARANCE'. Under 'SETTINGS', there are fields for 'Name' (set to 'SI'), 'Default time range' (set to 'Last 24 hours'), 'Dynamic Dashboard' (set to 'Static'), 'Width' (set to 'Auto'), and 'Date format' (set to '10/25/2022 09:57'). The bottom screenshot shows the 'Dashboards' list page. It includes a search bar ('Search Dashboard') and a filter for 'All organizations'. A newly created dashboard card for 'SI' is visible, along with other dashboard icons and a '+' button to create more.

Una vez que tenemos nuestro dashboard creado, accederemos al siguiente [artículo](#), y copiaremos el código en python expuesto en el.

Por lo que podemos ver en el código, este crea un nuevo dispositivo llamado "machine" con 3 variables: humedad, temperatura y ubicación.

Para conectarlo con nuestro dashboard en ubidots copiaremos nuestro "Token" y lo pegaremos en la variable "TOKEN"

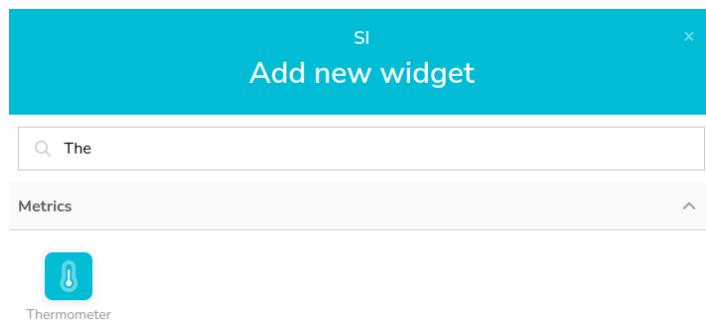
```
TOKEN = "...." # Put your TOKEN here
DEVICE_LABEL = "machine" # Put your device label here
VARIABLE_LABEL_1 = "temperature" # Put your first variable label here
VARIABLE_LABEL_2 = "humidity" # Put your second variable label here
VARIABLE_LABEL_3 = "position" # Put your second variable label here
```

Una vez hecho esto, ya podremos arrancar nuestro script, y veremos que empieza a emitir datos a ubidots.

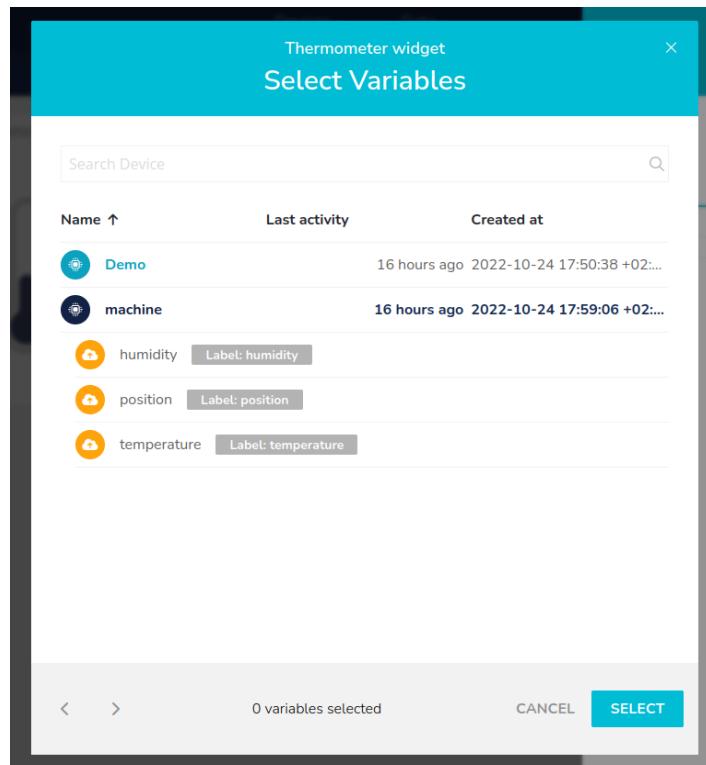
```
[INFO] request made properly, your device is updated
[INFO] finished
[INFO] Attempting to send data
200 {'humidity': [{'status_code': 201}], 'position': [{'status_code': 201}], 'temperature': [{'status_code': 201}]}
[INFO] request made properly, your device is updated
```

Ahora, en nuestro dashboard, podremos añadir widgets y enlazarlos automáticamente a las variables emitidas por el script.

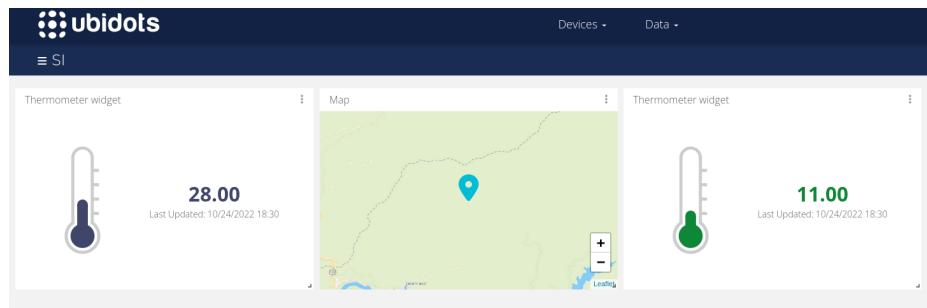
Para ello, volvemos a nuestro dashboard, y añadimos un par de widgets para mostrar la temperatura, la humedad, y un mapa para la ubicación.



Configuramos el widget, y le asociamos la variable.

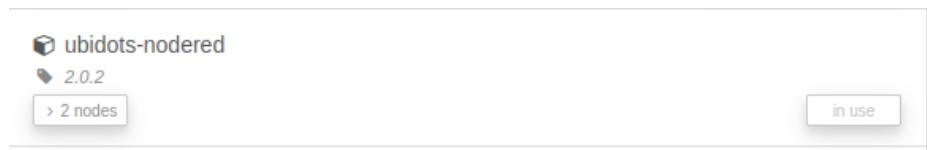


Añadiendo 2 termómetros, y el mapa, obtendremos un dashboard similar al siguiente:

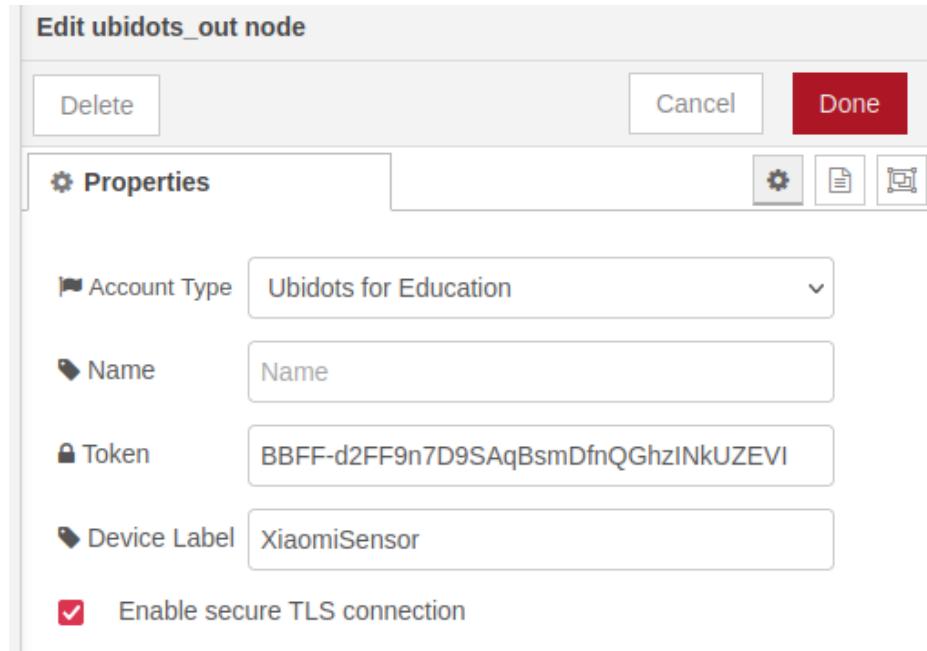


6.2.2 Ubidots usando node-red.

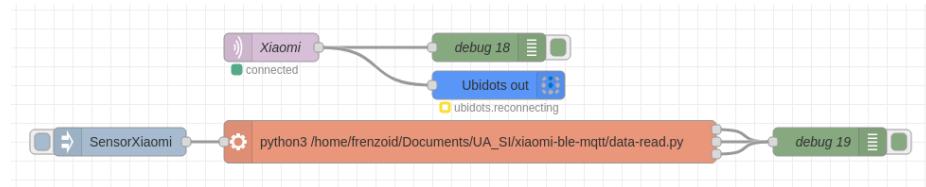
En node-red tenemos un nodo que nos permite interactuar con Ubidots.



Con este nodo (ubidots-out) podremos conectarnos a ubidots, y mandar datos a nuestro dashboard, la configuración permite especificar el tipo de cuenta, y el token a usar.



Hemos intentado conectarnos con ubidots usando este nodo, pero al final no lo logramos. El nodo de Ubidots no se conectaba con los servicios de Ubidots.

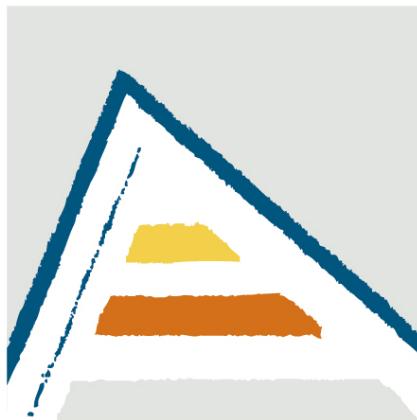


Pero en su defecto, siempre podemos usar un nodo http para atacar a la API Rest de Ubidots, y mandar los datos en formato JSON.

Sistemas Industriales

2022 / 2023

Especificación Técnica del Supuesto, propuesto por el grupo de Enfermería.



Universitat d'Alacant
Universidad de Alicante

- Elvi Mihai Sabau Sabau - 51254875L
- Daniel Asensi Roch - 48776120C
- Jéssica Román Marroquí - 74444117V
- María Verdú Belmonte - 77039404F
- Miguel Pérez Gómez - 74393992D
- Andrea Lillo - 48788941F
- Maria Soler - 29577771R
- Helena Rocamora Torres - 29525280L
- Maria Castell Navarro - 49230349E
- Ana Maria Martinez Salinas - 29528591H



Índice

1. Descripción del proyecto	2
2. Especificación Técnica	2
2.1. Servidor de entrenamiento	2
2.2. Servidor de Procesamiento	2
2.3. Servidor Web	3
2.4. Servidor de bases de datos	3
3. Diseño del sistema	4
3.1. Esquema de las relaciones de la base de datos	4
3.2. Esquema de la interconexión entre los módulos	5
4. Desarrollo del sistema	5
4.1. Metodología de desarrollo	5
4.2. Herramientas DevOps	6
4.3. Uso de servicios en Cloud (AWS)	7



1. Descripción del proyecto

El proyecto propuesto por el grupo de medicina es el de crear un sistema capaz de predecir la carga de trabajo para un turno de un hospital, estimar de forma automática la cantidad de personal necesario para lidiar con la carga de cada turno, gestionar los turnos de los enfermeros, ser capaz de intercambiar turnos, y que se pueda interactuar de manera cómoda desde una interfaz web o móvil.

Dada esta propuesta, podemos separar el sistema en 4 módulos principales.

- Un serviddor de entrenamiento: que entrenará un modelo de machine learning usando los datos proveídos por el grupo de enfermería, capaz de estimar el personal necesario dada cierta carga.
- Un servidor de procesamiento: capaz de gestionar los turnos de cada enfermero, que se encargue de recibir los datos de la interfaz, además de mostrar los turnos actuales y una previsión de los turnos.
- Un servidor web: que muestre una interfaz de los turnos, desde la cual los enfermeros puedan ver y solicitar cambios de turno, y que envíe dichas solicitudes a un servidor de procesamiento.
- Un servidor de bases de datos: donde guardaremos los turnos asignados a para enfermero, además de la carga estimada por el modelo por cada planta y turno, y de los registros de las solicitudes de los cambios de turno.

2. Especificación Técnica

La especificación técnica de cada módulo es el siguiente:

2.1. Servidor de entrenamiento

Para realizar el entrenamiento se usarán las siguientes tecnologías:

- **Tensorflow**: Usaremos tensorflow con Keras para entrenar un modelo basado en los datos proporcionados.
- **Python**: Usaremos el lenguaje de programación Python para interactuar con tensorflow, el propio lenguaje nos da una versatilidad que otros lenguajes no nos dan a la hora de manejar herramientas de machine learning.
- **Django**: Usaremos la librería Django para servir el modelo una vez entrenado, de esta manera nuestro modulo de procesamiento lo podrá descargar.

Además podremos continuar entrenando el modelo a medida que pasan los días dependiendo de lo buena o mala que sea la estimación.

2.2. Servidor de Procesamiento

Para realizar el procesamiento de las peticiones del servidor web, manejar los turnos, y conectar con la base de datos, se usarán las siguientes tecnologías:

- **Node**: Node es un entorno de ejecución de código JS, usaremos Node para ejecutar código JS en el servidor.
- **ExpressJS**: Express es una librería en JS que nos permite crear desde servidores web a APIRest, esto nos permiten recibir peticiones via HTTP de otros servicios, procesarlas, y responder a dichas peticiones. Además usaremos un middleware para interceptar las peticiones, y refrescar un JWT antes de responder a las peticiones, de esta manera podremos mantener un estado de sesión al interactuar con el front web (servidor web).



- **Socket.io:** SocketIO es una librería JS que nos permite crear conexiones en vivo mediante sockets usando el protocolo TCP/IP. Esta librería la usaremos para tener además una conexión en vivo con cada cliente web (front end), de esta manera podremos ver peticiones de solicitudes, cambios en los turnos o en la carga de trabajo en vivo, sin necesidad de recargar la página.
- **SequelizeORM:** Sequelize es una librería que hace de ORM entre nuestro programa y la base de datos a usar. Esta librería nos permite interactuar de manera fácil con la base de datos, el ORM mapea las tablas de la base de datos a clases de nuestro lenguaje, haciendo fácil la interacción con la base de datos.

2.3. Servidor Web

Usaremos las siguientes tecnologías para realizar la interfaz web.

- **NGINX:** NGINX es un servidor web que hostea tanto archivos planos en html como páginas dinámicas usando SSR. Este servidor además proporciona un servicio de balanceo de cargas, y un proxy para filtrar las conexiones.
- **Svelte:** es un framework de desarrollo de aplicaciones web progresivas en JS. Usaremos este framework por su facilidad y porque proporciona una capacidad de renderizado realmente reactivo.
- **Axios:** Usaremos Axios como gestor de peticiones HTTP, con el cual interactuaremos con el servidor de procesamiento (nuestro backend). Además nos beneficiaremos de su módulo de interceptadores para JWT como medida para mantener las sesiones activas en la página web.
- **Dragable:** Es una librería creada por shopify que nos permitirá añadir la interactuabilidad del arrastre entre elementos HTML, esta librería nos servirá para modelar nuestra página web, en específico el horario de los turnos.

2.4. Servidor de bases de datos

Usaremos en nuestro sistema 2 bases de datos.

- **PostgreSQL:** Este servidor base de datos la usaremos en conjunto con el servidor de procesamiento para guardar los turnos, sesiones, credenciales de los usuarios, plantas y otros. Hemos seleccionado esta base de datos porque nos permite la inserción masiva de datos, además de un amplio abanico de características, como (entre otros) la creación nativa de enumeradores.
- **Redis:** Este servidor base de datos la usaremos en conjunto con el servidor de entrenamiento para guardar en caché las estimaciones más recientes, y datos nuevos para entrenar actualizar y mejorar el modelo. Hemos seleccionado esta base de datos ya que nos permite guardar muchos datos de pequeño peso, además de guardarse en memoria (en RAM), haciéndola extremadamente rápida.

Además, podemos desplegar otra base de datos REDIS para cachear los resultados del servidor de procesamiento, y así aligerar nuestra APIRest en caso de que haya una alta carga de usuarios.

3. Diseño del sistema

En este apartado se detalla el esquema del Modelo Entidad Relación que se usará en la base de datos usada por el Back End.

También se muestra el esquema de interconexión entre los módulos a desarrollar.

3.1. Esquema de las relaciones de la base de datos

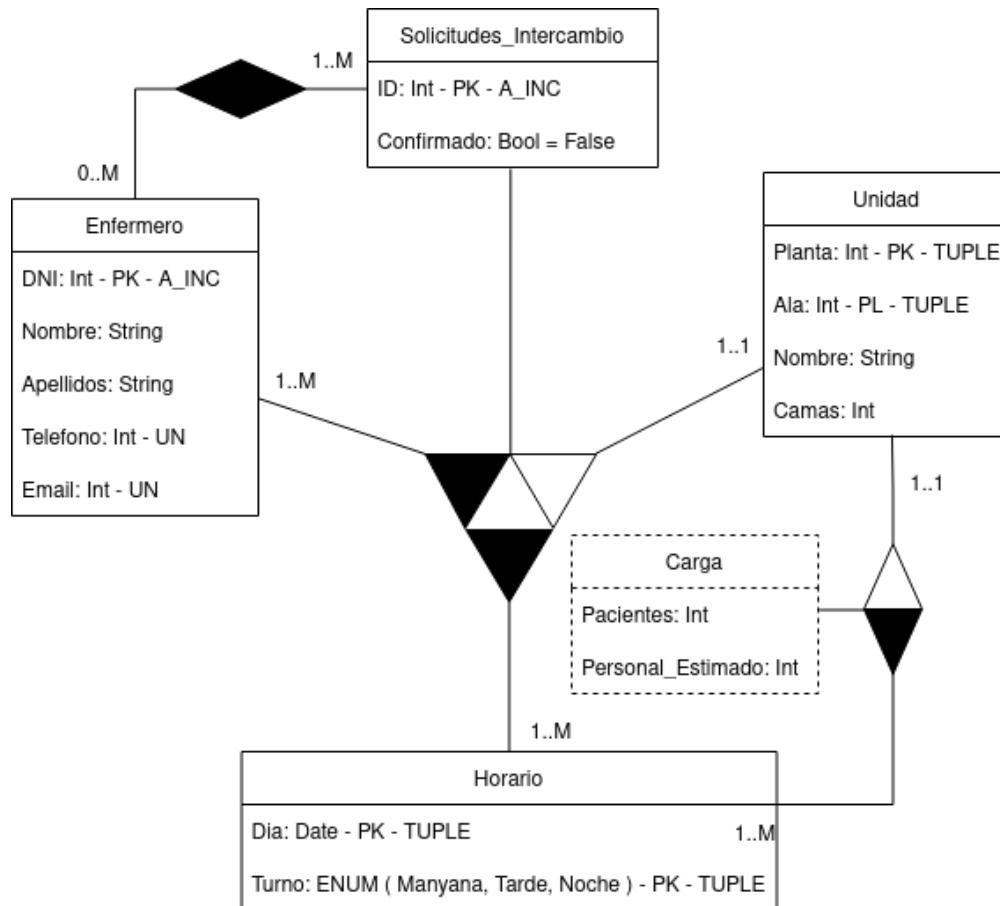


Figura 1: Diagrama EER de la base de datos (PostgreSQL).

La relación ternaria entre la unidad, el horario y el enfermero representa la relación restringida de uno o muchos enfermeros por cada planta en cada horario con su turno.

También la relación entre enfermero y *Solicitudes_Intercambio* representa la relación de cada enfermero al poder solicitar cambios de grupos a otros enfermeros de su misma unidad en un turno en concreto.

La carga es representada en el diagrama como un atributo de la relación entre el horario y la unidad, esta se aplica directamente al Horario, ya que dependiendo del turno en cada unidad (dependiendo de la cantidad de pacientes) se estima una cantidad de enfermeros necesarios.

3.2. Esquema de la interconexión entre los módulos

A continuación se muestra un pequeño esquema de interconexión entre los servicios descritos en los apartados anteriores.

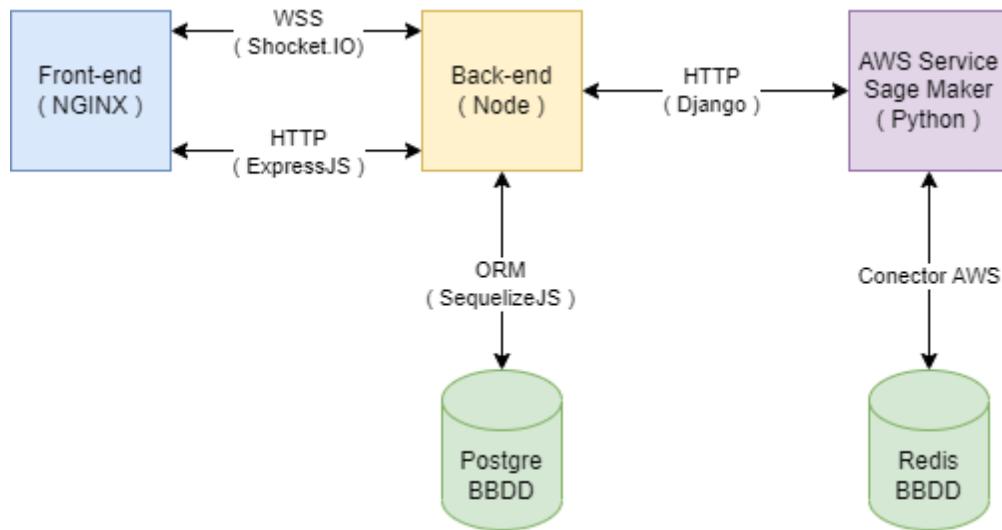


Figura 2: Esquema de interconexión entre servicios.

En este esquema de muestra la interconexión entre cada servicio y el protocolo que se usará para ello, además de la librería para dicha conexión. También especificamos el entorno / lenguaje de ejecución.

4. Desarrollo del sistema

A continuación describiremos la metodología, herramientas para el despliegue, y que servicios cloud que usaremos.

4.1. Metodología de desarrollo

La metodología a usar sera la de TDD (Test Driven Development), mediante integración continua usando Github Actions con Cicle.



4.2. Herramientas DevOps

Usaremos Docker para encapsular cada servicio y poder desplegarlo cómodamente. También usaremos [Portainer](#) para tener una interfaz sencilla a la hora de manejar nuestra instalación de Docker.

The screenshot shows the Portainer web interface. On the left is a sidebar with various navigation options: Home, LOCAL (Dashboard, App Templates, Stacks, Containers), Images, Networks, Volumes, Events, Host, SETTINGS (Users, Environments, Registries, Authentication logs, Settings). The 'Containers' option is currently selected. The main area is titled 'Container list' and shows a table of running containers. The table has columns for Name, State, Quick actions, Stack, and Image. The containers listed are: PORTAINER (running, portainer/portainer-ce:2.11.0), obsbpp_db_1 (running, obsbpp/mariadb:5), OB_MAIN_PAGE (running, 62d49f9bab67), ADMINER_INTERFACE (running, admirer:latest), obsbpp_sourcebans_1 (running, crinis/sourcebans:latest), TRAEFIK_CERTDUMPER (running, humenius/traefik-certs-dumper:latest), TRAEFIK (running, traefik:v2.0), MOSQUITTO_SERVER (stopped, eclipse-mosquitto:latest), MYSQL_DANIEL (stopped, mysql/mysql-server:5.7), MC_SERVER_2022 (stopped, itzg/minecraft-server:latest), and kafka (stopped, sdkafka/bitnami/kafka:latest).

Name	State	Quick actions	Stack	Image
PORTAINER	running	Start, Stop, Kill, Restart, Pause, Resume, Remove	-	portainer/portainer-ce:2.11.0
obsbpp_db_1	running	Start, Stop, Kill, Restart, Pause, Resume, Remove	obsbpp	mariadb:5
OB_MAIN_PAGE	running	Start, Stop, Kill, Restart, Pause, Resume, Remove	-	62d49f9bab67
ADMINER_INTERFACE	running	Start, Stop, Kill, Restart, Pause, Resume, Remove	-	admirer:latest
obsbpp_sourcebans_1	running	Start, Stop, Kill, Restart, Pause, Resume, Remove	-	crinis/sourcebans:latest
TRAEFIK_CERTDUMPER	running	Start, Stop, Kill, Restart, Pause, Resume, Remove	-	humenius/traefik-certs-dumper:latest
TRAEFIK	running	Start, Stop, Kill, Restart, Pause, Resume, Remove	-	traefik:v2.0
MOSQUITTO_SERVER	stopped	Start, Stop, Kill, Restart, Pause, Resume, Remove	-	eclipse-mosquitto:latest
MYSQL_DANIEL	stopped	Start, Stop, Kill, Restart, Pause, Resume, Remove	-	mysql/mysql-server:5.7
MC_SERVER_2022	stopped	Start, Stop, Kill, Restart, Pause, Resume, Remove	-	itzg/minecraft-server:latest
kafka	stopped	Start, Stop, Kill, Restart, Pause, Resume, Remove	sdkafka	bitnami/kafka:latest

Figura 3: Interfaz Portainer con varios contenedores en ejecución.

Y para gestionar los dominios de la página web y otros usaremos Traefik Proxy, certificando los dominios y renovándolos mediante Lets Encrypt. También podemos usar Digital Ocean como entidad certificadora.

4.3. Uso de servicios en Cloud (AWS)

Usaremos un servicio de Amazon Web Services - SageMaker para realizar el entrenamiento del modelo de la estimación de la carga de trabajo.

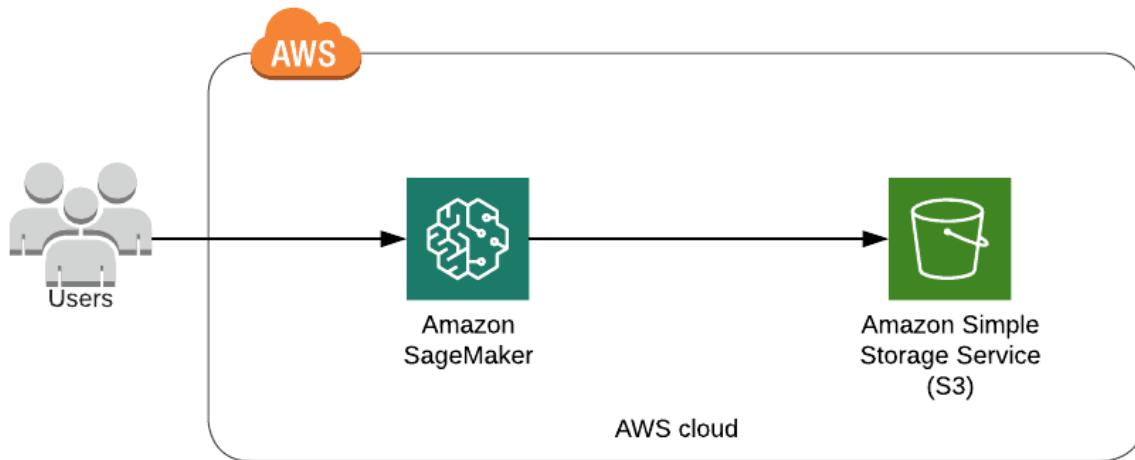


Figura 4: Esquema de interacción con AWS - SageMaker.

Y usaremos un servicio de AWS - Amazon Simple Storage Service mediante Redis para almacenar los datos más recientes en caché para el entrenamiento.