

# Cuarta práctica de SD

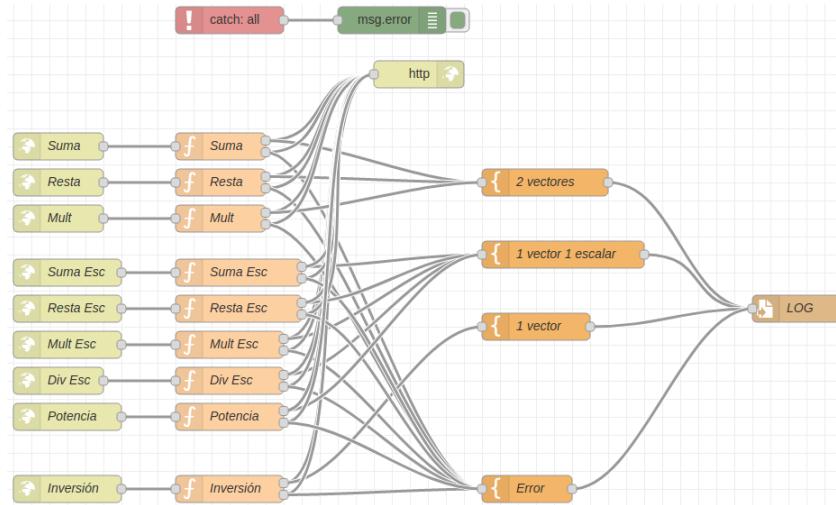
José Manuel Heredia Bravo  
Jesús Márquez Delgado

Mayo de 2020

## Índice

<b>1. Flujo 1: Calculadora de Vectores mediante API REST</b>	<b>2</b>
1.1. Operaciones con dos vectores . . . . .	2
1.1.1. Suma . . . . .	2
1.1.2. Resta . . . . .	3
1.1.3. Multiplicación . . . . .	4
1.2. Operaciones con un vector y un escalar . . . . .	4
1.2.1. Suma por un escalar . . . . .	5
1.2.2. Resta por un escalar . . . . .	5
1.2.3. Multiplicación por un escalar . . . . .	6
1.2.4. División por un escalar . . . . .	6
1.2.5. Elevar a un escalar . . . . .	7
1.3. Operación con un vector . . . . .	8
1.3.1. Invertir . . . . .	8
1.4. Registros . . . . .	9
1.4.1. Dos vectores y un resultado . . . . .	9
1.4.2. Un vector, un escalar y un resultado . . . . .	9
1.4.3. Un vector y un resultado . . . . .	9
1.4.4. Error . . . . .	9
1.5. Errores . . . . .	9
<b>2. Flujo 2: MQTT</b>	<b>10</b>
2.1. Suscripción con el protocolo MQTT . . . . .	10
2.2. Suscripción y publicación . . . . .	11
<b>3. Flujo 3: Contributed Nodes</b>	<b>12</b>
3.1. Nodos añadidos . . . . .	13
3.1.1. invert . . . . .	13
3.1.2. emboss . . . . .	13
3.1.3. brightness . . . . .	13

# 1. Flujo 1: Calculadora de Vectores mediante API REST



Se ha implementado una calculadora de vectores a través de Node-RED con los siguientes endpoints:

## 1.1. Operaciones con dos vectores

En cada uno de los casos se comprueba que los elementos de ambos vectores están compuestos por elementos numéricos.

### 1.1.1. Suma

- URL: GET /suma
- Recibe dos vectores numéricos y devuelve un vector con los elementos de ambos vectores sumados.
- Implementación:

```
1 msg.data = {"tipo" : "Suma"};
2 if(a.length < b.length){
3     for(i = a.length; i < b.length; i++)
4         a[i] = 0;
5 }
6 else{
7     for(i = b.length; i < a.length; i++)
8         b[i] = 0;
9 }
10 for(i = 0; i < a.length; i++){
```

```

11     if(isNaN(a[i])){
12         nan = "a contiene elementos no numéricos";
13         node.error(nan,msg);
14         msg.payload = {"error":nan};
15         return [null,msg];
16     }
17     if(isNaN(b[i])){
18         nan = "b contiene elementos no numéricos";
19         node.error(nan,msg);
20         msg.payload = {"error":nan};
21         return [null,msg];
22     }
23     c[i] = a[i] + b[i];
24 }
25 msg.payload = { "resultado" : c};
26 return [msg,null];

```

---

### 1.1.2. Resta

- URL: GET /resta
- Recibe dos vectores numéricos y devuelve un vector con los elementos de ambos vectores restados.
- Implementación:

---

```

1 msg.data = {"tipo" : "Resta"};
2 if(a.length < b.length){
3     for(i = a.length; i < b.length; i++)
4         a[i] = 0;
5 }
6 else{
7     for(i = b.length; i < a.length; i++)
8         b[i] = 0;
9 }
10 for( i = 0; i < a.length; i++){
11     if(isNaN(a[i])){
12         var nan = "a contiene elementos no numéricos";
13         node.error(nan,msg);
14         msg.payload = {"error":nan};
15         return [null,msg];
16     }
17     if(isNaN(b[i])){
18         var nan = "b contiene elementos no numéricos";
19         node.error(nan,msg);
20         msg.payload = {"error":nan};
21         return [null,msg];

```

```

22     }
23     c[i] = a[i] - b[i];
24   }
25 msg.payload = { "resultado" : c};
26 return [msg,null];

```

---

### 1.1.3. Multiplicación

- URL: GET /mult
- Recibe dos vectores numéricos y devuelve la suma de la multiplicación de los elementos de los vectores en la misma posición, como ambos vectores deben tener el mismo tamaño, se enviará un mensaje de error si se da dicho caso.
- Implementación:

```

1 msg.data = {"tipo" : "Multiplicación"};
2 if(a.length !== b.length){
3   nan = "Ambos vectores deben tener el mismo numero de elementos";
4   node.error(nan,msg);
5   msg.payload = {"error":nan};
6   return [null,msg];
7 }
8 for(i = 0; i < a.length; i++){
9   if(isNaN(a[i])){
10     nan = "a contiene elementos no numéricos";
11     node.error(nan,msg);
12     msg.payload = {"error":nan};
13     return [null,msg];
14   }
15   if(isNaN(b[i])){
16     nan = "b contiene elementos no numéricos";
17     node.error(nan,msg);
18     msg.payload = {"error":nan};
19     return [null,msg];
20   }
21   res += a[i] * b[i];
22 }
23 msg.payload = { "resultado" : res};
24 return [msg,null];

```

---

## 1.2. Operaciones con un vector y un escalar

En cada uno de los casos se comprobará que el vector solo contiene elementos numéricos y además que el escalar sea un número también.

### 1.2.1. Suma por un escalar

- URL: GET /sumaeesc
- Recibe un vector numérico y devuelve el vector con todos sus elementos sumados por el escalar.
- Implementación:

---

```
1 msg.data = {"tipo" : "Suma Escalar"};
2 if(isNaN(n)){
3     nan = "n no es numéricos";
4     node.error(nan,msg);
5     msg.payload = {"error":nan};
6     return [null,msg];
7 }
8 if(n !== 0){
9     for(var i = 0; i < a.length; i++){
10         if(isNaN(a[i])){
11             nan = "a contiene elementos no numéricos";
12             node.error(nan,msg);
13             msg.payload = {"error":nan};
14             return [null,msg];
15         }
16         res[i] = a[i] + n;
17     }
18 }
19 msg.payload = { "resultado" : res};
20 return [msg,null];
```

---

### 1.2.2. Resta por un escalar

- URL: GET /restaesc
- Recibe un vector numérico y devuelve el vector con todos sus elementos restados por el escalar.
- Implementación:

---

```
1 msg.data = {"tipo" : "Resta Escalar"};
2 if(isNaN(n)){
3     nan = "n no es numéricos";
4     node.error(nan,msg);
5     msg.payload = {"error":nan};
6     return [null,msg];
7 }
8 if(n !== 0){
```

```

9     for(var i = 0; i < a.length; i++){
10         if(isNaN(a[i])){
11             nan = "a contiene elementos no numéricos";
12             node.error(nan,msg);
13             msg.payload = {"error":nan};
14             return [null,msg];
15         }
16         res[i] = a[i] - n;
17     }
18 }
19 msg.payload = { "resultado" : res};
20 return [msg,null];

```

---

### 1.2.3. Multiplicación por un escalar

- URL: GET /multesc
- Recibe un vector numérico y devuelve el vector con todos sus elementos multiplicados por el escalar.
- Implementación:

```

1 msg.data = {"tipo" : "Suma Escalar"};
2 if(isNaN(n)){
3     nan = "n no es numéricos";
4     node.error(nan,msg);
5     msg.payload = {"error":nan};
6     return [null,msg];
7 }
8 if(n !== 0){
9     for(var i = 0; i < a.length; i++){
10         if(isNaN(a[i])){
11             nan = "a contiene elementos no numéricos";
12             node.error(nan,msg);
13             msg.payload = {"error":nan};
14             return [null,msg];
15         }
16         res[i] = a[i] + n;
17     }
18 }
19 msg.payload = { "resultado" : res};
20 return [msg,null];

```

---

### 1.2.4. División por un escalar

- URL: GET /divesc

- Recibe un vector numérico y devuelve el vector con todos sus elementos divididos por el escalar, si el escalar es un 0, enviará un mensaje de error.
- Implementación:

---

```

1 msg.data = {"tipo" : "División Escalar"};
2 if(isNaN(n)){
3     nan = "n no es numéricos";
4     node.error(nan,msg);
5     msg.payload = {"error":nan};
6     return [null,msg];
7 }
8 if(n === 0){
9     nan = "Division por 0";
10    node.error(nan,msg);
11    msg.payload = {"error":nan};
12    return [null,msg];
13 }
14 else{
15     if(n !== 0){
16         for(var i = 0; i < a.length; i++){
17             if(isNaN(a[i])){
18                 nan = "a contiene elementos no numéricos";
19                 node.error(nan,msg);
20                 msg.payload = {"error":nan};
21                 return [null,msg];
22             }
23             res[i] = a[i] / n;
24         }
25     }
26     msg.payload = { "resultado" : res}
27 }
28 return [msg,null];

```

---

#### 1.2.5. Elevar a un escalar

- URL: GET /pow
- Recibe un vector numérico y devuelve el vector con todos sus elementos elevados por el escalar.
- Implementación:

---

```

1 msg.data = {"tipo" : "Potencia"};
2 if(isNaN(n)){
3     nan = "n no es numéricos";
4     node.error(nan,msg);
5     msg.payload = {"error":nan};
6     return [null,msg];
7 }
8 if(n !== 0){
9     for(var i = 0; i < a.length; i++){
10         if(isNaN(a[i])){
11             nan = "a contiene elementos no numéricos";
12             node.error(nan,msg);
13             msg.payload = {"error":nan};
14             return [null,msg];
15         }
16         res[i] = Math.pow(a[i],n);
17     }
18 }
19 msg.payload = { "resultado" : res};
20 return [msg,null];

```

---

### 1.3. Operación con un vector

Se comprobará si todos los elementos del vector son numéricos y en caso negativo se enviará un mensaje de error.

#### 1.3.1. Invertir

- URL: GET /inver
- Recibe un vector numérico y devuelve el vector con todos sus elementos con su signo invertido.
- Implementación:

```

msg.data = {"tipo" : "Inversión"};
for(var i = 0; i < a.length; i++){
    if(isNaN(a[i])){
        nan = "a contiene elementos no numéricos";
        node.error(nan,msg);
        msg.payload = {"error":nan};
        return [null,msg];
    }
    a[i] *= -1;
}
msg.payload = { "resultado" : a};
return [msg,null];

```

## 1.4. Registros

Los registros de las operaciones recibidas se almacenarán en un archivo log en el directorio del usuario con diferentes plantillas dependiendo del tipo de operación.

### 1.4.1. Dos vectores y un resultado

---

```
1 {{data.tipo}} a: ({{req.body.a}}) b: ({{req.body.b}})
2 resultado: ({{payload.resultado}})
```

---

Ejemplo: Suma a: (1,2,3) b: (4,5,6) resultado: (5,7,9)

### 1.4.2. Un vector, un escalar y un resultado

---

```
1 {{data.tipo}} a: ({{req.body.a}}) n: ({{req.body.n}})
2 resultado: ({{payload.resultado}})
```

---

Ejemplo: Suma Escalar a: (1,2,3) n: (2) resultado: (3,4,5)

### 1.4.3. Un vector y un resultado

---

```
1 {{data.tipo}} a: ({{req.body.a}}) resultado: ({{payload.resultado}})
```

---

Ejemplo: Inversión a: (-1,-2,-3) resultado: (-1,-2,-3)

### 1.4.4. Error

---

```
1 {{data.tipo}} Error: ({{payload.error}})
```

---

Ejemplo: Resta Escalar Error: (n no es numéricos)

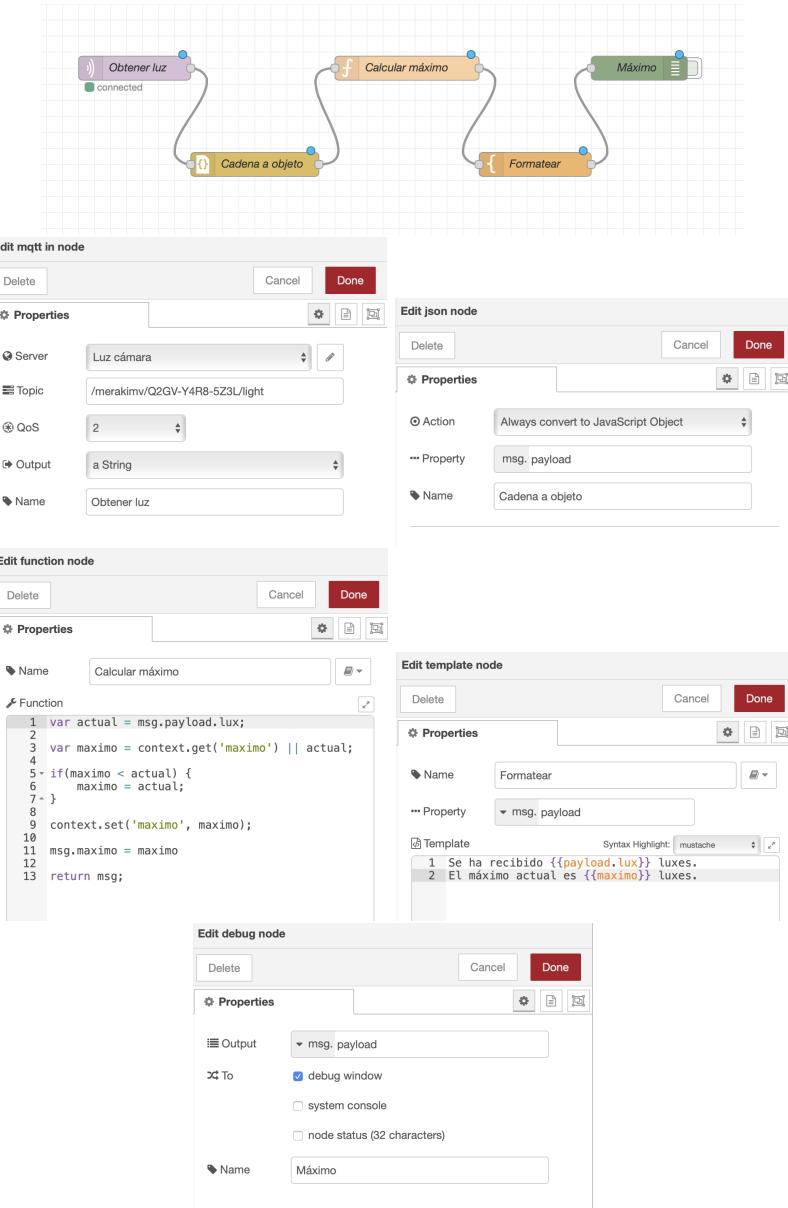
## 1.5. Errores

Los errores detectados por el programa, además de ser enviado al usuario y ser almacenado en el fichero log, se mostrará un mensaje por la consola de debug, enviado a través del nodo catch.

## 2. Flujo 2: MQTT

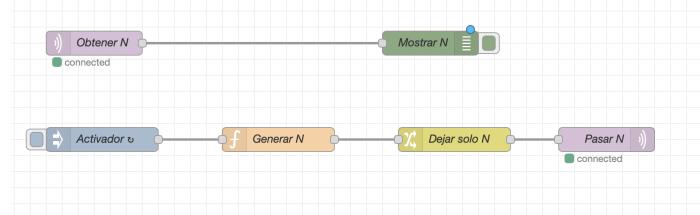
### 2.1. Suscripción con el protocolo MQTT

El flujo va a ser el siguiente:



## 2.2. Suscripción y publicación

Ahora vamos a crear, en dos flujos distintos, el comportamiento de un suscriptor y el de un publicador.



Cada nodo significa:

- **Activador.** Cada cierto tiempo (dos segundos), manda una marca de tiempo. Esta información es innecesaria en un principio, pero lo utilizaremos como activador del nodo función a continuación explicado.
- **Generar N.** Devuelve un número aleatorio entre 0 y 100.
- **Dejar solo N.** Elimina del payload la marca de tiempo (es innecesaria).
- **Pasar N.** Pasa el payload (en este caso, el número aleatorio n).

En la parte del suscriptor, lo único que hará es recibir dicho número aleatorio y lo mostrará por la consola de depuración.

Las configuraciones de los nodos son las siguientes:

**Edit inject node**

Delete Cancel Done

Properties

Payload timestamp

Topic

Inject once after 1 seconds, then

Repeat interval

every 2 seconds

Name Activador

Note: "interval between times" and "at a specific time" will use cron.  
"interval" should be 596 hours or less.  
See info box for details.

**Edit function node**

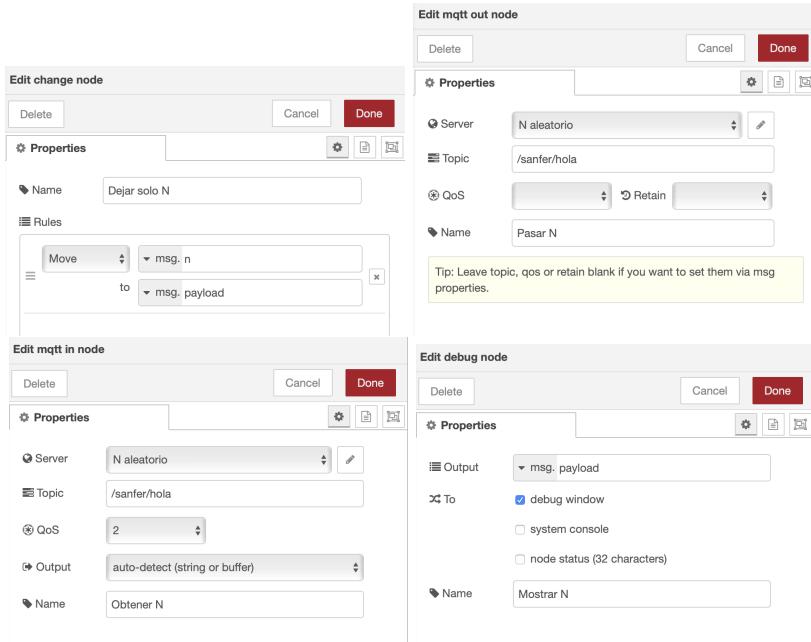
Delete Cancel Done

Properties

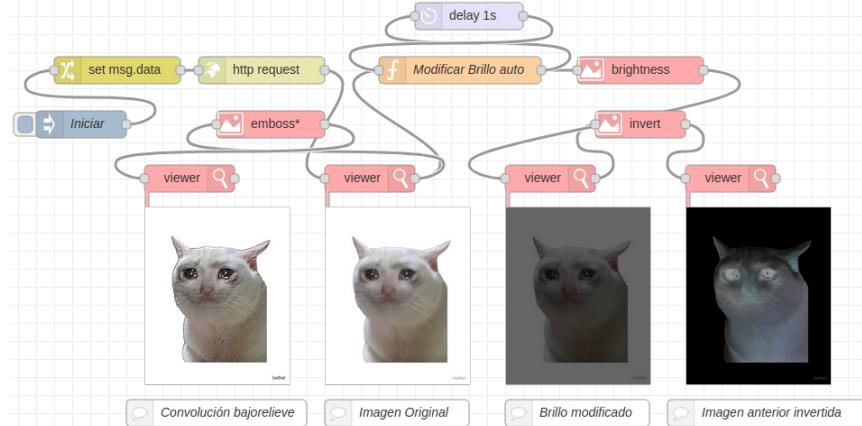
Name Generar N

Function

```
1 msg.n = Math.floor(Math.random()*101);
2
3 return msg;
```



### 3. Flujo 3: Contributed Nodes



Para este ejercicio se ha instalado [node-red-contrib-image-tools](#) el cual nos permite realizar fácilmente tratamiento de imágenes.

Se han elegido estos nodos para comprobar la simpleza de realizar estas operaciones utilizando node-red.

### 3.1. Nodos añadidos

La instalación nos añade un nodo de “viewer”, el cual nos muestra la imagen la cual le envíemos, un nodo “image”, el cual tiene múltiples operaciones las cuales pueden aplicarse a una imagen cargada y un nodo “Barcode Decoder” el cual decodifica los códigos de barras, qr ... .

Para esta prueba hemos utilizado el nodo viewer y algunas de las operaciones que nos ofrece image.

#### 3.1.1. invert

Invierte los colores de la imagen que recibe.

#### 3.1.2. emboss

Aplica una convolución a la imagen para darle un efecto de relieve, la operación “emboss” trae una matriz para realizar la convolución ya cargada pero “image” también trae una operación “convolute” la cual realiza una convolución a partir de una matriz dada.

#### 3.1.3. brightness

Modifica el brillo de la imagen dada a través de un valor recibido entre 1 y -1, para realizar la prueba se ha creado un bucle con un retardo de 1s el cual va aumentando el nivel de brillo desde -1 a 1 aumentado en 0.1, una vez llega a 1 se reinicia el valor a -1 de nuevo.

---

```
1 msg.data += 0.1;
2
3 if(msg.data >= 1)
4     msg.data = -1;
5
6 return msg;
```

---