

Internet of Things - Third Challenge

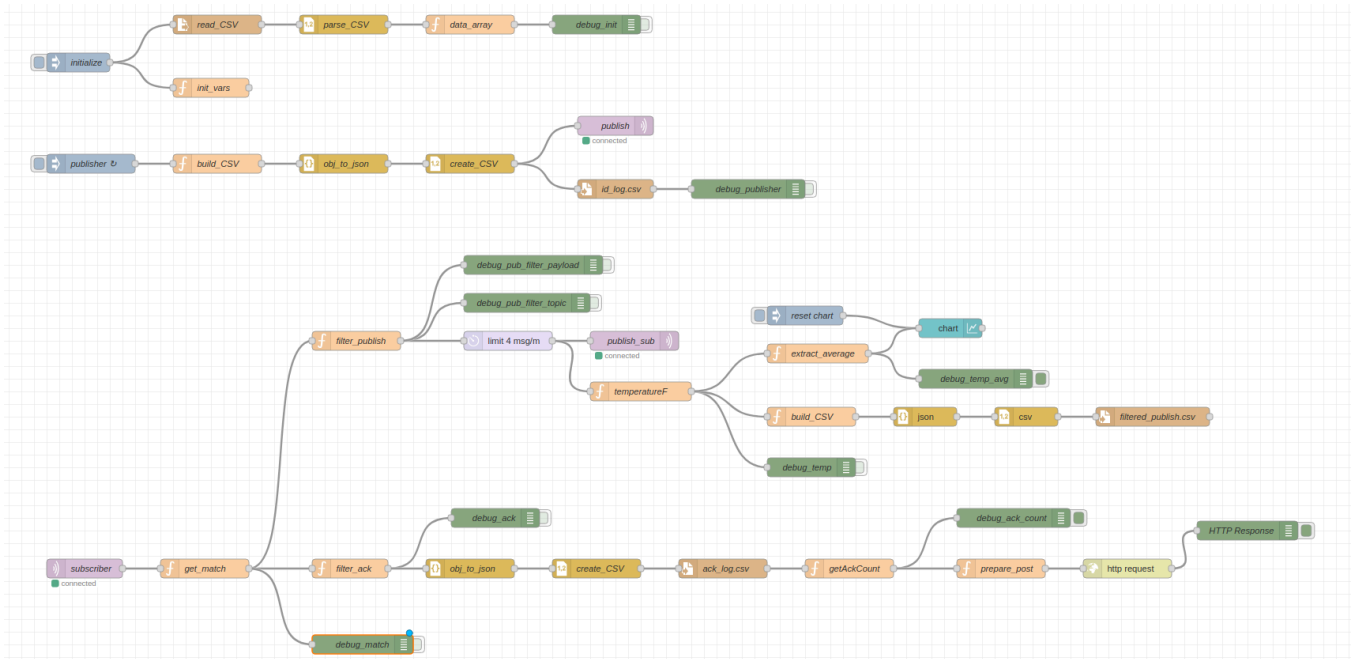
Authors:

- Abate Kevin: **10812892**
- Pigato Lorenzo: **10766953** [Team Leader]

Thingspeak channel ID: <https://thingspeak.mathworks.com/channels/2929989>

Project topic

The project consists in building some **Node-Red** flows:



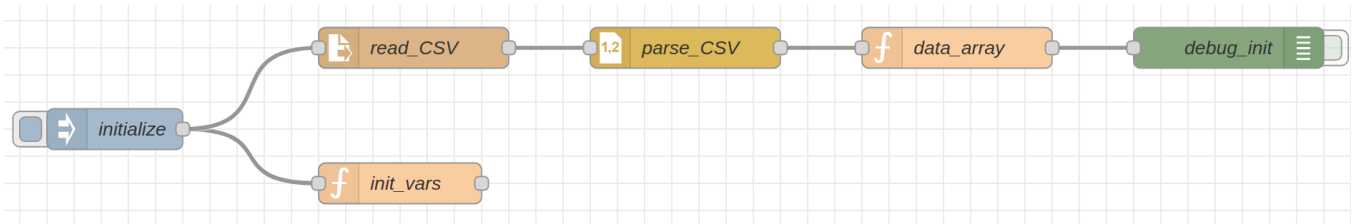
First Part

Periodically publish MQTT messages to the local mosquitto broker (localhost, port 1884), to the topic *challenge3/id_generator*.

Messages should be sent with a rate of 1 message every 5 seconds.

Each message should contain in the payload a string of JSON format with a random number (id) between 0 and 30000, and the time in which the msg is generated (UNIX timestamp)

Initialize



- **Initialized:** start global variable initialization
- **init_vars:** initialize global variables

```
global.set("Status", "FLOWING");
global.set("row_num", 1)
global.set("ack_count", 0);
global.set("msg_count", 0);
global.set("row_temp", 1);

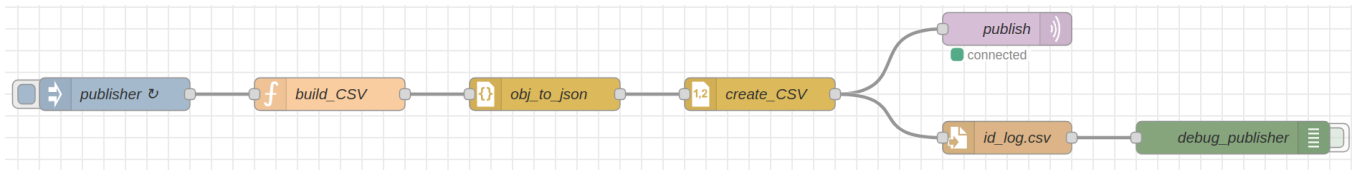
return msg;
```

- ****read_csv**:** read the `_challenge3.csv_` file
- ****parse_csv**:** parse the `_.csv_` file
- ****data_array**:** declare a global variable with all file's data

```
```javascript
var data = msg.payload;
global.set("csv_data",data);

return msg;
```

## Publisher



- **publisher**: publish with a rate of 1 message every 5 second to activate the flow
- **build\_csv**: prepares the payload of the MQTT message and *id\_log.csv* file:

```
let row_num = global.get("row_num");

// Define the ID field as a random integer between 0 and 30000
const id = Math.floor(Math.random() * 30001);
const timestamp = Math.floor(Date.now() / 1000);

// Set the message payload
msg.payload = JSON.stringify({
 "No.": row_num,
 "ID": id,
 "TIMESTAMP": timestamp
});

// Progressively increases the row number
row_num++;
global.set("row_num", row_num);

return msg;
```

- **obj\_to\_json**: translate the object message into JSON format
  - **create\_csv**: use JSON input to produce a line for a .csv file
  - **publish**: send a MQTT message with defined payload on fixed topic *challenge3/id\_generator*
  - **id\_log.csv**: append the payload to the *id\_log.csv* file
-

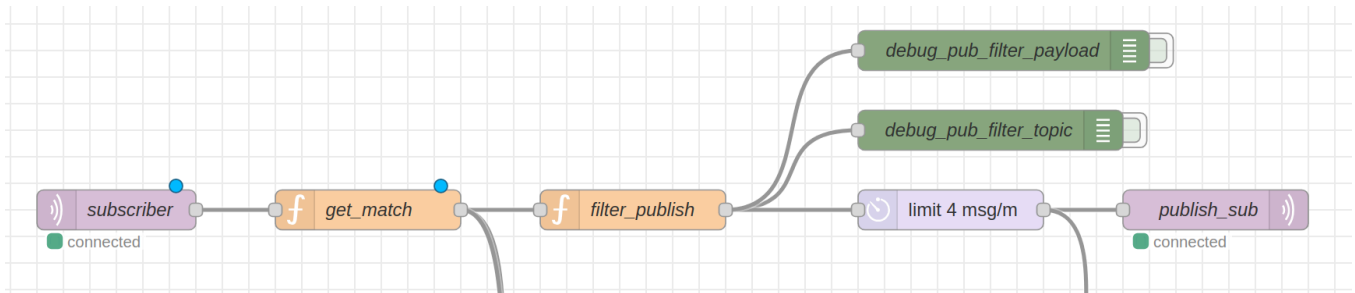
## Second Part

Subscribe to the topic *challenge3/id\_generator* in the local broker.

After receiving a message from the subscription, take the ID and compute  $N = ID \% 7711$

At every received message, process the challenge3.csv file and take the message with frame number equal to N.

If the message contains an MQTT Publish then, publish a message on the same topic found with.



- **subscriber**: connects to the broker and subscribes to messages from the specified topic
- **get\_match**: finds the line of the data which "No." matches with the calculated "N" and produces a node message object with its payload. This node feeds multiple branches:

```
let msg_count = global.get("msg_count");

// No more than 80 csv entries has to be processed
if(msg_count >= 80){
 global.set("Status", "COMPLETE");
 return null;
}

var fields = msg.payload.split(',');

var id = fields[1];
var data = global.get("csv_data");

// Compute N (computed_id)
var computed_id = id % 7711;
var matching_obj = data.filter(
 function(obj){
 return parseInt(obj["No."])===computed_id;
 }
);

msg.payload = matching_obj;

msg_count++;
global.set("msg_count", msg_count);

return msg;
```

- **filter\_publish**: filter to transmit forward only the messages containing a MQTT Publish. If a single message contains more than a publish, then separate the payload into different messages:

```

if (global.get("Status") === "COMPLETE") {
 return;
}

let matching_obj = msg.payload;
let row_num = matching_obj[0]["No."];
let info = matching_obj[0]["Info"];
let rawPayload = matching_obj[0]["Payload"] || "";

if (matching_obj && info.includes("Publish Message")) {

 // Use a regex to find all the topics [../../..]
 let topic_regex = /\[([^\]]+)\]/g;
 let topics = [... info.matchAll(topic_regex)].map(match => match[1]);

 // Parsing
 let payloads;
 try {
 // Normalize the payload removing ""
 let normalized = "[" + rawPayload.replace(/"/g, "'") + "]";
 payloads = JSON.parse(normalized);
 } catch (err) {
 node.error("Error while parsing JSON: " + err.message);
 return null;
 }

 // Match each topic and message
 let output_msgs = [];
 for (let i = 0; i < Math.min(topics.length, payloads.length); i++) {
 output_msgs.push({
 topic: topics[i],
 payload: {
 timestamp: Math.floor(Date.now() / 1000),
 id: row_num,
 payload: payloads[i]
 }
 });
 }

 return [output_msgs]; // output
} else {
 return null;
}

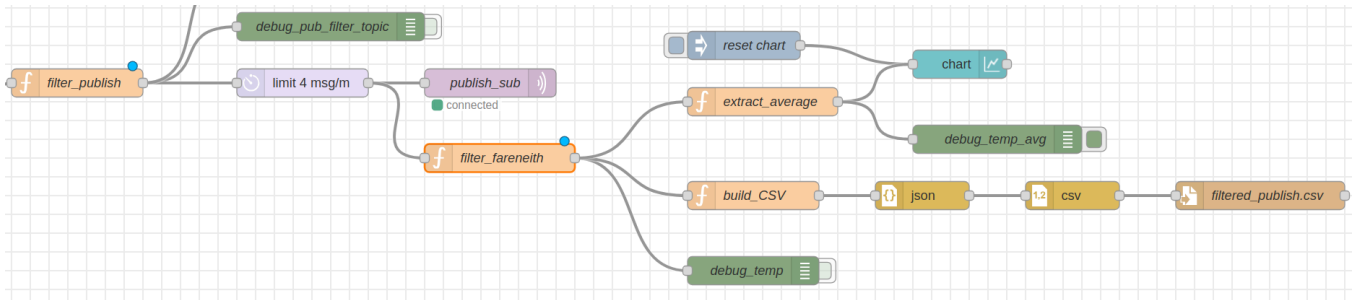
```

- **limit 4msg/m**: limit message rate to 4 per minute, add messages exceeding the rate to a queue
  - **publish\_sub**: publish the payload as a MQTT message on topic defined by .csv file
-

## Third part

If the filtered message contains in the payload a temperature in Fahrenheit, use it to produce a chart in Node-Red plotting the temperature value, taking the mean value of the "range" attribute in the payload.

Save the payload of these msgs in a CSV (filtered\_pubs.csv)



- **filter\_fareneith**: filter to forward only messages with `type: temperature, unit: F` :

```
let input = msg.payload;

if (!input || !input.payload) {
 return null;
}

let data = input.payload;

if (data.type === "temperature" && data.unit === "F") {
 return {
 topic: msg.topic,
 payload: input
 };
}

return null;
```

- **extract\_average**: extracts the average temperature from the payload of the message:

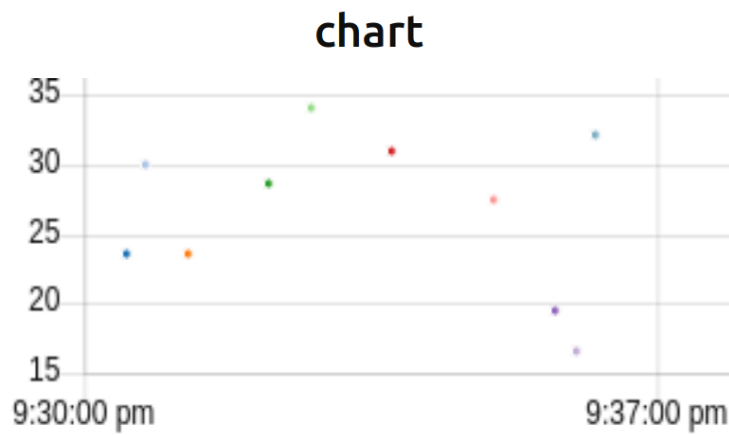
```
let data = msg.payload.payload;

let sum = data.range.reduce((a, b) => a + b, 0);
let avg = sum / 2;

msg.payload = avg;

return msg;
```

- **chart**: plot the average contained inside the payload onto a chart:



- **build\_csv**: prepares the payload for the *filtered\_publish.csv* file:

```
var unit = data.unit;
var description = data.description;

let avg = (range[0] + range[1]) / 2;

msg.payload = JSON.stringify({
 "No.": row_temp,
 "LAT": lat,
 "LONG": long,
 "RANGE": avg,
 "TYPE": type,
 "UNIT": unit,
 "DESCRIPTION": description,
});

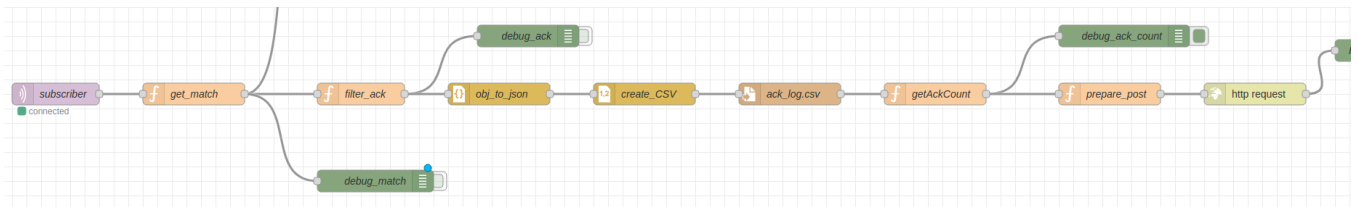
row_temp++;
global.set("row_temp", row_temp);
return msg;
```

- **json**: translate the object message into JSON format
  - **csv**: use JSON input to produce a line for a .csv file
  - **filtered\_publish.csv**: append the payload to the *filtered\_publish.csv* file
-

## Fourth part

If the message with Frame No. = N instead contains an MQTT ACK message, increment a global ACK counter, then save the message into a CSV file named *ack\_log.csv*

Send the value of the global ACK counter to a *thingspeak* channel, passing to the field1 of the channel the value of the global ACK counter



- **filter\_ack**: filter to forward only messages containing an ACK:

```
if(global.get("Status")==="COMPLETE"){
 return;
}

var ack_count = global.get("ack_count");
var matching_obj = msg.payload;
var row_num = matching_obj[0]["No."];

if(matching_obj && matching_obj[0]["Info"].includes("Ack")){
 var ack_regex = /(Connect|Publish|Subscribe|Unsubscribe)\sAck/;
 var matching_msg_info = matching_obj[0]["Info"].match(ack_regex);

 //Use JSON.stringify to convert string in right format
 msg.payload = JSON.stringify({
 "TIMESTAMP": Math.floor(Date.now() / 1000),
 "SUB_ID": row_num,
 "MSG_TYPE": matching_msg_info[0]
 });

 // Increment acq counter
 global.set("ack_count", ack_count +1);

 return msg;
} else {
 return null;
}
```

- **obj\_to\_json**: translate the object message into JSON format
- **create\_csv**: use JSON input to produce a line for a .csv file
- **ack\_log.csv**: append the payload to the *ack\_log.csv* file
- **get\_ack\_count**: extract ack counter from payload:

```
msg.payload = global.get("ack_count");
return msg;
```

- **prepare\_post**: setup the payload to be published as an HTTP POST:

```
let apiKey = "S7CY4USHQ42EZI0S";

msg.payload = `api_key=${apiKey}&field1=${msg.payload}`;

msg.headers = {
 "Content-Type": "application/x-www-form-urlencoded"
};
```



```
return msg;
```

- **http request:** send the payload as an HTTP POST

The graph visualized on Thingspeaks:

