POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

IOT Challenge 3

Author: **Matteo Leonardo Luraghi**

Person Code: 10772886
Academic Year: 2024-25

Contents

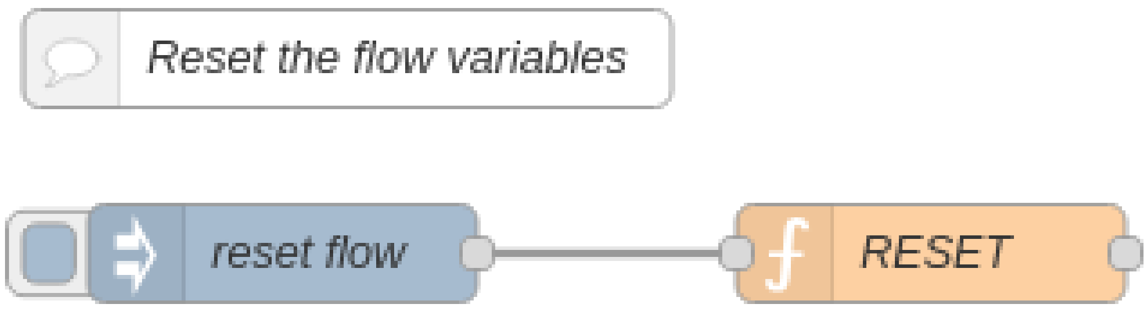
1 | Node-RED

1.1. Flow

My Node-RED flow consists of three distinct branches, each responsible for executing a sequence of actions.

The following sections describe each block in detail, excluding the debug blocks.

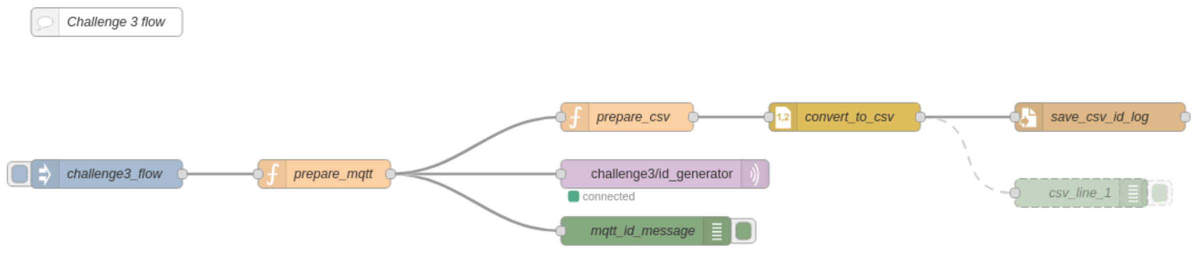
1.1.1. Reset Branch



This branch is responsible for resetting all flow variables used within the application. It ensures a clean state, manually triggered when needed, the flow variables are:

- `message_number`: counter of MQTT messages generated with a random ID between 0 and 30000
- `N`: mqtt message ID modulo 7711 as specified in point 2 of the documentation
- `limiter`: counter that tracks the 80 messages to process. After 80 messages, the LIMITER object will block any further messages from passing through the branch
- `temperature_number`: number of messages containing a temperature in Fahrenheit processed from the challenge3.csv file
- `ack_counter`: number of ACK messages processed from the challenge3.csv file and sent via the HTTP API to ThingSpeak

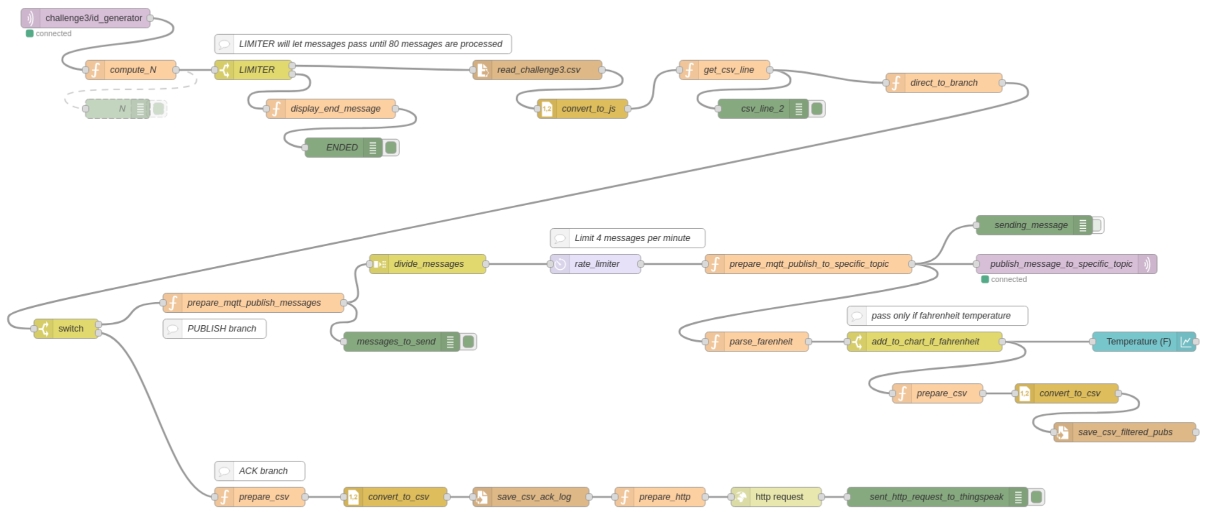
1.1.2. Publisher Branch



This branch sends a message every 5 seconds to the mosquitto broker hosted locally on port 1884, the components of this branch are:

- `challenge3_flow`: injects a new message every 5 seconds to trigger the branch, running indefinitely until the flow is stopped
- `prepare_mqtt`: function block that generates a random ID between 0 and 30000, sets the mqtt message topic to "challenge3/id_generator" and sets the mqtt message payload as {"id": ID, "timestamp": CURRENT_TIMESTAMP}
- `challenge3/id_generator`: mqtt out block that publishes the message to the "challenge3/id_generator" topic on the local mosquitto broker running on port 1884
- `prepare_csv`: function block that initialises to 1 if needed and updates the flow variable `message_number` and then creates the CSV line in the format [No., ID, TIMESTAMP] to be appended to the id_log.csv file
- `convert_to_csv`: csv block that converts the message data into a CSV line
- `save_csv_id_log`: write file block that appends the CSV line generated by `convert_to_csv` to the id_log.csv file

1.1.3. Subscriber branch



This branch connects to the same local mosquitto broker and subscribes to the topic "challenge3/id_generator". Upon receiving a message, the data is processed accordingly.

The components of this branch are:

- `challenge3/id_generator`: mqtt in block that subscribes to the topic "challenge3/id_generator" on the mosquitto broker (localhost, port 1884)
- `compute_N`: function block that initialises the flow variable `limiter` to 0 if needed, increments it, and computes the flow variable `N` as the modulo of the received mqtt message ID and 7711
- `LIMITER`: switch block that allows messages to proceed if and only if the flow variable `limiter` is less than or equal to 80, ensuring no more than 80 messages are processed
- `display_end_message`: function block that sets the message payload to `"LIMIT REACHED"` to be displayed in the debug console if the `LIMITER` block redirects the message to this sub-branch
- `read_challenge3.csv`: read file block that parses the challenge3.csv file
- `convert_to_js`: csv block that converts the lines of the challenge3.csv file to a javascript array
- `get_csv_line`: function block that uses the flow variable `N` to retrieve the corresponding line from the challenge3.csv file
- `direct_to_branch`: function block that parses the message payload's `Info` field to determine whether the message is a "Publish Message", an "Ack" message, or

another type. It then sets the `msg.branch` field to direct the message to the correct sub-branch (`PUBLISH`, `ACK`, or `IGNORED`)

- `switch`: switch block that routes the message based on the `msg.branch` field to the appropriate sub-branch

Below is a detailed description of the sub-branches relative to the `PUBLISH` and `ACK` messages.

PUBLISH branch:

- `prepare_mqtt_publish_messages`: function block that processes an incoming message to extract topics from the "Publish Message" field, creates a list of payloads with necessary details (timestamp, id, topic, and payload), and stores the results in the `msg.messages_to_send` field
- `divide_messages`: split block that takes the array of payloads from `msg.messages_to_send` (created in the `prepare_mqtt_publish_messages` block) and divides them into individual messages, which are then passed to the `rate_limiter` block
- `rate_limiter`: delay block that restricts the flow to a maximum of 4 messages per minute
- `prepare_mqtt_publish_to_specific_topic`: function block that sets the `msg.topic` field to the current message's topic, assigns the entire message to `msg.payload`, and updates `msg.payload.timestamp` with the current timestamp
- `publish_message_to_specific_topic`: mqtt out block that publishes the message to the topic specified in the message itself on the local mosquitto broker running on port 1884
- `parse_farenheiht`: function block that attempts to parse the message payload. If the message contains a Fahrenheit temperature, it sets the `msg.temp_check` flag to `true`, stores the temperature data, and calculates the average temperature to be used in the chart. If the payload doesn't contain a valid Fahrenheit temperature, the function sets `msg.temp_check` to `false` and clears the payload
- `add_to_chart_if_farenheit`: switch block that allows messages to proceed if and only if the `msg.temp_check` field is set to true
- `Temperature (F)`: chart block that takes the temperature input and displays it alongside the previous values
- `prepare_csv`: function block that initialises to 1 if needed and updates the flow

variable `temperature_number` and then creates the CSV line that will later be appended to the filtered_pubs.csv file in the format
[No., LONG, LAT, MEAN_VALUE, TYPE, UNIT, DESCRIPTION]

- `convert_to_csv`: csv block that converts the message to a CSV line
- `save_csv_filtered_pubs`: write file block that appends the CSV line generated by texttttconvert_to_csv to the filtered_pubs.csv file

ACK branch:

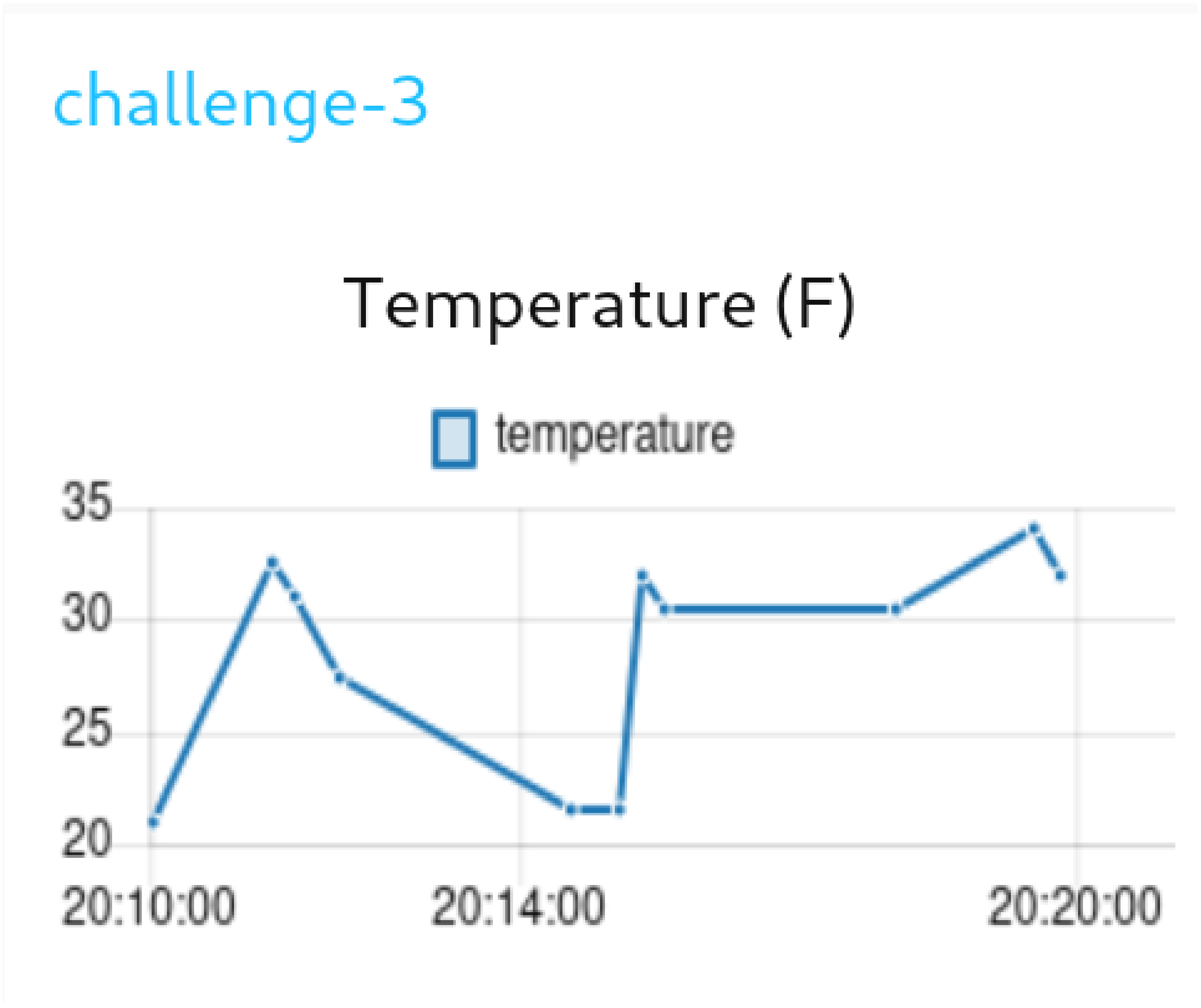
- `prepare_csv`: function block that initialises the flow variable `message_number` to 0 if needed, increments it and then creates the CSV line that will later be appended to the ack_log.csv file in the format [No., TIMESTAMP, SUB_ID, MSG_TYPE]
- `convert_to_csv`: csv block that converts the message to a CSV line
- `save_csv_ack_log`: write file block that appends the CSV line generated by `convert_to_csv` to the ack_log.csv file
- `prepare_http`: function block that sets `msg.method` to `"GET"` and `msg.url` to the appropriate endpoint, including the API key, to send a GET request to ThingSpeak for publishing the updated ACK counter
- `http request`: http request block that sends the http request to ThingSpeak

1.2. Flow Execution

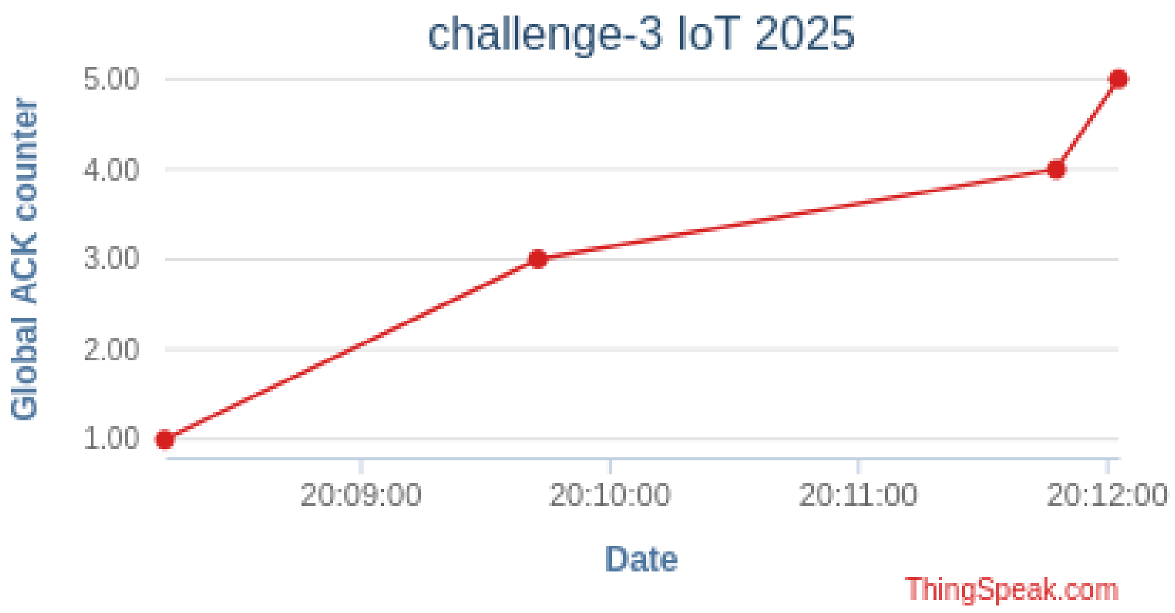
The flow was initiated by starting the generation of messages every 5 seconds.
Once the limit of 80 processed messages was reached, the flow continued to process any remaining messages queued in the rate limiter.
After all queued messages were processed, the flow was stopped.
After the processing, the ThingSpeak API key was changed, so the one found in the nodered.txt file will not work.

1.3. Charts

1.3.1. Node Red Chart



1.3.2. ThingSpeak Chart



ThingSpeak Channel