Introduction

The third IoT challenge posed a scenario that required a practical application of Internet of Things (IoT) concepts using Node-RED, a flow-based development tool designed for visual programming and connecting hardware devices, APIs, and online services in new and interesting ways. The task revolved around simulating an environment where IoT devices publish messages containing sensor data, represented by a randomly generated ID and a timestamp.

Node-RED's capabilities to handle data flow, process information, and communicate with MQTT protocols were put to the test. MQTT, a lightweight messaging protocol for small sensors and mobile devices, was essential for this challenge, as it was used to publish and subscribe to topics within the local network broker.

The overarching objective of the challenge involved several sub-tasks:

1. Set up an MQTT environment to periodically publish messages containing a unique identifier (ID) and a timestamp.
2. Ensure the implementation of a storage system to log these messages into a CSV file, creating a persistent record of the data flow.
3. Subscribe to the same MQTT topic to retrieve these messages and process them to extract and compute specific data points based on given conditions.
4. Engage with a provided CSV file (challenge3.csv) that simulated the sensor data messages which needed to be analyzed and filtered based on the processed ID values from the MQTT messages.
5. Identify and handle MQTT publish messages that contain temperature readings, specifically in Fahrenheit, and visualize this data on a Node-RED dashboard using a graphical chart.
6. Track MQTT acknowledgement messages and interact with an external service, ThingSpeak, to report on these occurrences.

To meet these demands, Node-RED nodes were used to create a series of operations that transformed and routed data. This process encompassed the generation of messages, processing based on content, logging in various formats, conditional handling, and external API interaction, providing a comprehensive demonstration of Node-RED's IoT data processing capabilities.

The challenge not only assessed the ability to utilize Node-RED effectively but also the understanding of MQTT's role in IoT for message handling and the integration of cloud services (ThingSpeak) to achieve a full-cycle data flow, reflecting real-world IoT scenarios.

This report will detail the implementation steps and the logic behind each Node-RED node used to construct the flow, ensuring clarity and insight into how each task within the challenge was addressed and executed.

MQTT Message Generation and Logging

This part of the Node-RED flow is dedicated to simulating an IoT device that generates data, akin to a sensor in a real-world scenario. Here's a breakdown of each node's role and how they collectively fulfill the requirements for this portion of the challenge:

1. **Inject Node**: The flow's initiation point; it's configured to automatically trigger at deployment and subsequently every 5 seconds. In a typical IoT setup, this would simulate a sensor or a device sending data at regular intervals. The payload output by this node is the current timestamp, which simulates the time at which a sensor reading was taken.

2. **Function Node (function 1)**: Acts as the "sensor" that generates the data. It produces a random ID number between 0 and 50,000 to mimic a unique identifier that an IoT device might send. Along with the ID, the current UNIX timestamp from the **inject** node's payload is attached to create a JSON object. This mirrors how a sensor would package its data (with a unique ID and timestamp) before transmission.

3. **JSON Node**: The JSON string conversion here is crucial for MQTT communication, as MQTT payloads are commonly formatted as JSON strings. This ensures the message is structured in a way that is expected by both the publisher and subscribers within an MQTT network.

4. **MQTT Out Node**: Represents the transmitter of the IoT device, publishing the generated JSON string to the MQTT broker on the **challenge3/id_generator** topic. The local MQTT broker, specified in the challenge, is the central hub that handles these messages.

5. **Function Node (function 2)**: This node receives the JSON object, extracts the ID and timestamp, and prepends a sequence number (**No.**). The sequence number serves as a running count of messages sent, mimicking a log file's entries where each new message is given an incremental identifier to maintain order and for easy reference.

6. **CSV Node**: Converts the object containing **No.**, **ID**, and **TIMESTAMP** into a CSV formatted string. CSV (Comma-Separated Values) is a widely used data storage format that is easy to read for both humans and machines. It's especially useful for logging data in a tabular form.

7. **File Out Node**: Acts as the storage mechanism. It takes the CSV string from the **csv** node and appends it to a file called **id_log.csv**. This file path is set to a user-specified directory within the Node-RED environment. The node is configured not to overwrite the existing file but to append new lines, thereby creating a continuous log of published messages over time.

By implementing these steps, we've created a system that mimics an IoT sensor generating and transmitting data, and logs this data for record-keeping, debugging, and further analysis. This setup covers both real-time data handling via MQTT and persistent storage via CSV logging, providing a foundation for more complex IoT data processing tasks.

MQTT Message Subscription and Processing

This segment of the Node-RED workflow focuses on the reception and subsequent handling of MQTT messages, which would typically be the responsibility of a server or another IoT device within the network.

1. **MQTT In Node**: This node is effectively the listener in the MQTT communication model. It subscribes to the **challenge3/id_generator** topic, waiting for messages published to this channel. Once a message is received, the node outputs it for further processing. This simulates an IoT device or a service that reacts to incoming sensor data.
2. **Function Node (function 9)**: Here, the message flow is controlled to prevent processing more messages than required by the challenge. This function node maintains a count of how many messages have been processed and, after receiving the 80th message, will halt the flow of additional messages. In practice, this could be used to manage resource consumption or limit processing to a set data batch.
3. **Function Node (function 3)**: After a message pass through the control mechanism, this function node takes the ID from the message payload and performs a modulo operation with the specified number (7711), which yields **N**. This **N** value is used to link the MQTT message with a specific entry in a CSV file, representing how a real-world application might need to process and relate incoming data to existing records or datasets.
4. **File In Node**: The challenge specifies a **challenge3.csv** file that contains predefined data. This node reads the entire contents of the file, simulating access to a database or another data store in response to an event (in this case, the reception of an MQTT message).
5. **CSV Node**: The CSV file's contents are then parsed into a JavaScript array of objects, where each object represents a row from the file. This conversion is essential for programmatically accessing the data within the flow. Given that CSV is a common export format for data logging in IoT applications, this node demonstrates how Node-RED can ingest such data for processing.

Conditional Processing Based on Payload Content

After obtaining the required row from the CSV data, further processing depends on the content of the payload:

1. **Function Node (function 4)**: It searches through the parsed CSV data to find the row where the **Frame No.** matches the previously computed **N**. Once the matching row is found, the payload of that specific row is prepared for the next steps in the flow.
2. **Switch Node**: Functions as a decision-making point in the flow. It evaluates the payload to determine if the message contains either a "Publish Message" or an "ACK" message. This routing is analogous to an application's logic layer, where decisions are made based on data content, directing the flow to appropriate processing functions.

The subsequent actions after the **Switch Node** vary depending on the message type:

- For "Publish Message" payloads, the flow branches handle the rate limiting, message publishing back to the MQTT topic, temperature data extraction, chart plotting, and logging temperature data into a CSV.
- For "ACK" payloads, the flow increments a global counter, logs the ACK message, and sends the updated counter to the ThingSpeak channel via an HTTP request.

This part of the flow encapsulates the ability of Node-RED to subscribe to MQTT topics, process messages, and perform actions based on specific conditions. It demonstrates how Node-RED can serve as a middle layer in an IoT stack, linking incoming data streams with both internal processing logic and external services or endpoints.

Final Steps and Cleanup

The latter stages of the Node-RED flow are designed to gracefully complete the challenge's requirements by ensuring proper handling of all conditions specified, as well as cleaning up and preparing the system for a potential restart or next round of data processing.

1. **Switch Node (after function 9)**: This node is a crucial part of the flow control; it is set up to detect when the maximum number of messages (80) specified by the challenge has been received. Upon receiving the 80th message, it sends the flow to a **debug** node which can be used to notify the operator that the limit has been reached. All further messages are ignored, which prevents any further processing or logging, effectively halting the challenge as per the requirements.
2. **Debug Nodes (debug 10, debug 12, debug 13)**: These are strategically placed throughout the flow for troubleshooting and monitoring purposes. When active, they provide real-time feedback on the data passing through the flow at their respective points, which can be invaluable for debugging or when checking that the flow is operating as expected.
3. **Function Node (Extract Temperature F)**: This node performs a specialized task of extracting temperature values from the payload only if the temperature is in Fahrenheit. It is part of the data processing branch that handles "Publish Message" payloads containing temperature readings. The function node also averages the min and max values of the temperature range provided and then formats the data for both logging and chart plotting.
4. **CSV Node (after Extract Temperature F)**: Takes the temperature data formatted by the "Extract Temperature F" function node and converts it into a CSV string. This node is configured to write the CSV headers once, ensuring the resulting CSV file (**filtered_pubs.csv**) is properly formatted for future reference or analysis.
5. **File Node (Log Temperature Data)**: Logs the temperature data into **filtered_pubs.csv**. Like earlier file nodes, it appends data to the file, creating a record of all temperature readings processed during the challenge.
6. **UI Chart Node (Temperature Chart)**: Used to plot the temperature data on a Node-RED dashboard. This visual representation could be used to monitor temperature readings over

time, which in a real-world scenario could provide insights into trends or trigger alerts if temperatures go beyond certain thresholds.

7. **Function Node (function 10)**: This node, positioned after the "Publish Message" section of the **Switch** node, is intended to prepare the messages for MQTT re-publishing and rate limiting. It handles messages individually, creating a new message object that includes the current timestamp, the ID from the original message, and the payload extracted from the CSV file.

8. **Delay Node**: Ensures that messages are published at a controlled rate, in this case, four messages per minute as specified by the challenge. It is a form of traffic shaping to avoid overwhelming the MQTT broker or any downstream systems that consume these messages.

9. **HTTP Request Node (in combination with function 8)**: This is the last node in the ACK message processing branch and is used to send the value of the global ACK counter to ThingSpeak. This integration showcases Node-RED's capability to interact with web services and APIs, which is a common requirement in IoT applications for actions like reporting data to cloud services or triggering web-based workflows.

The final steps and cleanup portion of the flow ensure that the challenge's constraints are respected, and that the relevant data is logged and presented as required. The flow ceases further processing after the defined limit, preventing unnecessary resource usage and potential errors from processing excess data. The use of debug nodes provides visibility into the flow's operation, which is essential for both development and production monitoring. Additionally, the use of the charting function highlights Node-RED's capability to not only process data but also to serve as a dashboarding and monitoring tool.

Deliverables

The deliverables for IoT Challenge #3 provide a comprehensive overview of the work completed during the challenge. They serve as artifacts of the process and results, and they enable a review of the approach and outcomes. Here is an expanded explanation of each deliverable:

1. **Node-RED Flow Export as JSON (ch3(1).docx)**: This JSON file represents the entire Node-RED flow used to solve the challenge. It is an export of all the nodes, their configurations, and their interconnections, allowing the flow to be imported and replicated in another Node-RED instance. This document serves as a blueprint of the solution and is fundamental for sharing, reviewing, or iterating on the developed flow.

2. **CSV Files Produced**: Three CSV files (**id_log.csv**, **filtered_pubs.csv**, and **ack_log.csv**) are the data outputs of the flow, acting as logs for the different activities that took place during the challenge:

    - **id_log.csv** records every generated ID and timestamp pair, showing the continuous data stream that was simulated.

    - **filtered_pubs.csv** contains records of temperature data extracted from payloads, providing a focused dataset for temperature-related events.

- **ack_log.csv** keeps track of all acknowledgment messages processed, which is useful for auditing and understanding the flow of control messages within the MQTT protocol.

3. **ThingSpeak Channel Update**: **[ThingSpeak Channel ID: 6APLRLFIGEA8DZOI]** The global ACK counter value is sent to a ThingSpeak channel, demonstrating the ability to integrate with cloud services and APIs. ThingSpeak is an IoT analytics platform service that allows aggregating, visualizing, and analyzing live data streams in the cloud. Providing the link to this channel offers a real-time view of the counter's status and validates the flow's external communication capabilities.

4. **Screenshot of the Node-RED Flow (Screenshot 2024-04-20 230413.png)**: A visual representation of the flow in Node-RED gives a snapshot of how the nodes are arranged and connected, offering a quick understanding of the flow's structure and logic briefly.