University of L'Aquila

Information Engineering Computer Science and Mathematics

UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA

Software Engineering for Internet of Things

Project Report
Hydroponic System Monitoring

Team Members

Abu Saleh

Aruzhan Amangeldina

Tesfay W. Tesfay

Submitted To

Professor Davide Di Ruscio

L'Aquila, 20 January 2026

Contents

1 Introduction

The increasing demand for efficient and sustainable agricultural practices has accelerated the adoption of hydroponic farming systems. Hydroponics enables precise control of environmental and water-related parameters, leading to improved crop yield, reduced resource consumption, and suitability for controlled or urban environments. However, maintaining optimal conditions requires continuous monitoring of multiple variables, making manual supervision inefficient and unreliable.

Advances in the Internet of Things (IoT) and autonomous systems provide effective solutions to these challenges by enabling real-time data collection, automated monitoring, and timely alerts. Through the integration of sensors, communication protocols, and data processing platforms, hydroponic systems can operate with minimal human intervention while maintaining system stability and responsiveness.

This project presents a Smart Hydroponics IoT System developed as part of the Software Engineering for the Internet of Things course. The system focuses on software-driven autonomy by simulating sensor data, monitoring environmental conditions in real time, and notifying users when critical thresholds are exceeded. The report details the objectives, requirements, technologies, and architecture of the system, demonstrating the application of software engineering principles to an autonomous IoT-based agricultural solution.

2 Objectives

The primary objective of this project is to design and implement an IoT-based monitoring system for hydroponic farming environments using modern software engineering principles. The system aims to provide continuous visibility into critical environmental parameters while minimizing the need for manual supervision. The specific objectives of the project are:

- To simulate hydroponic sensor data representing key environmental and water-quality parameters such as temperature, humidity, pH, EC, and light intensity.
- To enable reliable, real-time data communication using a lightweight publish–subscribe messaging protocol.
- To process incoming sensor data and evaluate it against predefined thresholds in an automated manner.
- To store sensor readings efficiently for historical analysis and monitoring.
- To visualize real-time and historical data through interactive dashboards.
- To provide instant alert notifications when monitored parameters exceed safe operating limits.
- To design a modular and containerized system that is easy to deploy, maintain, and extend.

These objectives collectively support the development of a scalable and autonomous monitoring solution suitable for smart hydroponic farming scenarios.

3 Functional Requirements

Functional requirements describe the specific behaviors and capabilities that the system must provide in order to fulfill its intended purpose. They define what the system should do in terms of data collection, processing, storage, visualization, and alerting.

Table 1: Functional Requirements

ID	Name	Description
FR-1	Sensor Simulation	The system simulates hydroponic sensors including air temperature, humidity, water pH, water temperature, electrical conductivity, and light intensity using a Python-based simulator. Sensor values follow realistic ranges, evolve over time, and each sensor is uniquely identified by location, sensor type, and sensor ID.
FR-2	Communication Protocol	The system uses MQTT as the main communication protocol. Sensor data is published to structured topics following the format `/agriculture/<location>/<sensor_type>/<sensor_id>`
FR-3	Data Ingestion	The system subscribes to MQTT topics and receives sensor data streams in real time.
FR-4	Data Processing	The system processes incoming sensor messages, extracts measurement values and metadata, and prepares them for storage and analysis.
FR-5	Autonomous Evaluation	The system autonomously evaluates sensor values against predefined threshold ranges without human intervention.
FR-6	Alert Triggering	The system automatically detects abnormal conditions and triggers alert events when thresholds are violated.
FR-7	Notification Delivery	The system sends alert notifications to users through a Telegram bot, including sensor type, value, and location.
FR-8	Data Storage	The system stores time-stamped sensor data in a time-series database for historical analysis.
FR-9	Data Visualization	The system visualizes real-time and historical sensor data using interactive dashboards.
FR-10	Configuration Management	The system allows threshold values, credentials, and connection parameters to be configured through environment variables.

4 Non-Functional Requirements

Non-functional requirements define the quality attributes and constraints of the system rather than specific behaviors. They describe how the system should perform, focusing on aspects such as performance, reliability, scalability, security, and maintainability.

Table 2: Non-Functional Requirements

ID	Name	Description
NFR-1	Portability	The system is fully containerized using Docker Compose, ensuring consistent deployment and execution across different environments and machines.
NFR-2	Reliability	The system operates continuously and processes incoming sensor data without data loss during normal operation.
NFR-3	Scalability	The system supports the addition of new sensors, topics, and monitored parameters without requiring architectural changes.
NFR-4	Real-Time Responsiveness	The system processes sensor data and triggers alerts with minimal delay to enable timely reaction to critical conditions.
NFR-5	Maintainability	The system is modular, with clearly separated services, allowing components to be updated or replaced independently.
NFR-6	Usability	The system provides clear and intuitive dashboards for monitoring sensor values and system status.
NFR-7	Security	The system externalizes credentials and configuration parameters using environment variables to prevent hard-coded sensitive data.

5 System Architecture

The Smart Hydroponics IoT System is designed as a modular, autonomous monitoring solution that integrates sensor simulation, real-time data processing, storage, visualization, and alerting. The system follows a layered architecture that separates concerns between data generation, communication, processing, storage, and user interaction.

5.1 Architecture Overview

This section describes the main technologies and tools used to design and implement the Smart Hydroponics IoT System architecture, and explains the role of each component within the overall architecture.

5.1.1 Python (Sensor Simulator)

Python is used to implement the hydroponic sensor simulator responsible for generating synthetic sensor data. The simulator produces realistic values for air temperature, humidity, water pH, water

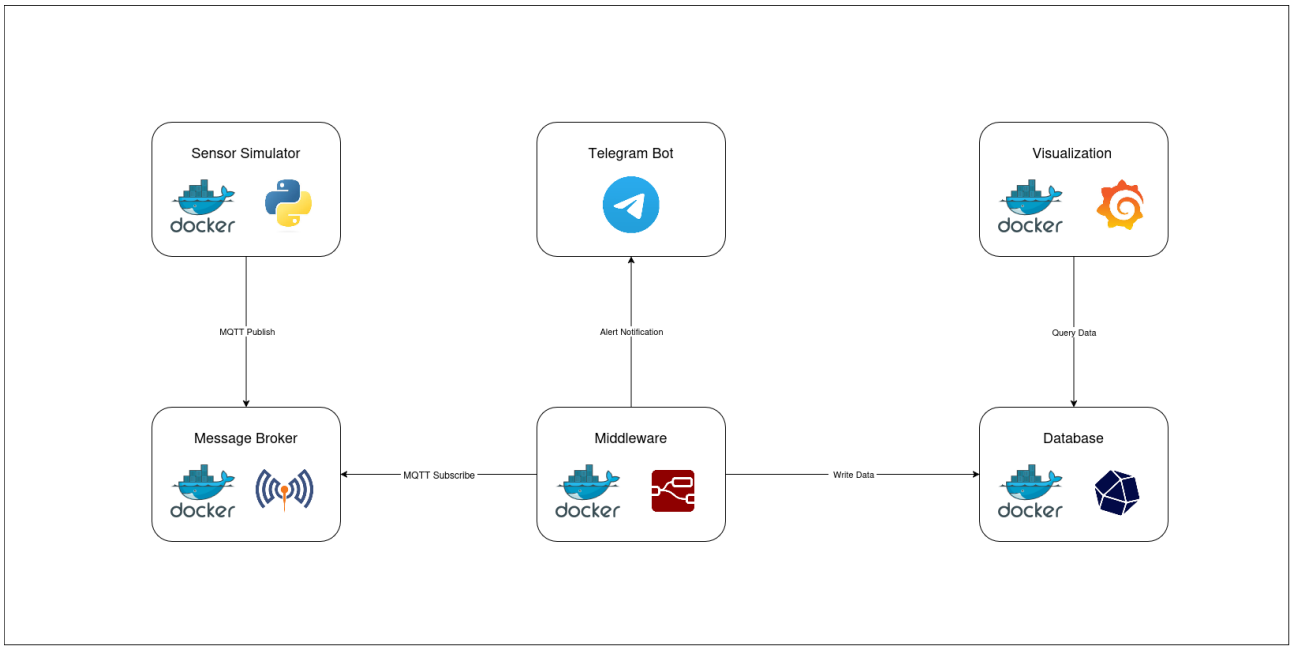

Figure 1: System Architecture

temperature, electrical conductivity (EC), and light intensity. Sensor values evolve over time within predefined ranges to emulate real hydroponic conditions. The simulator is configurable through environment variables, allowing the number of sensors, locations, and publishing intervals to be adjusted without modifying the source code. The paho-mqtt library is used as an MQTT client to publish sensor measurements to the MQTT broker using structured topics of the form:

5.1.2 Mosquitto (MQTT Broker)

Mosquitto is used as the MQTT broker for message exchange within the system. It enables lightweight, reliable, and asynchronous communication between the simulated hydroponic sensors and downstream components. The publish–subscribe model decouples data producers from consumers, improving scalability and fault tolerance. All system components subscribe to relevant MQTT topics to receive sensor data in real time.

5.1.3 Node-RED (Data Processing and Automation)

Node-RED serves as the middleware layer responsible for processing incoming sensor data and implementing autonomous system logic. It subscribes to MQTT topics, parses sensor payloads, and evaluates measurements against predefined threshold values. Node-RED implements rule-based automation to detect abnormal conditions, such as out-of-range pH levels or excessive temperature, and triggers corresponding actions without human intervention.

5.1.4 InfluxDB (Time-Series Database)

InfluxDB is used as the time-series database for storing all hydroponic sensor measurements. It is optimized for high write throughput and efficient querying of time-stamped data, making it suitable for continuous IoT data streams. Each measurement is stored together with timestamps

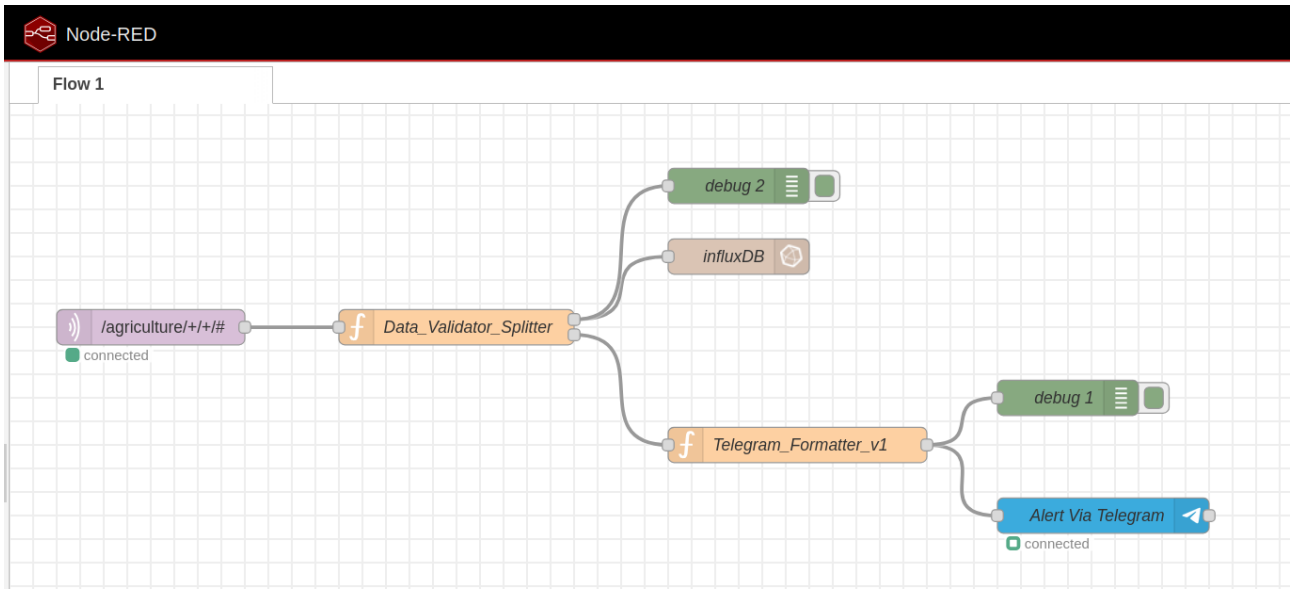

Figure 2: Node-Red Automation Flow

and tags such as location, sensor type, and sensor ID, enabling both real-time monitoring and historical analysis.

5.1.5 Grafana (Visualization Dashboard)

Grafana is used to visualize sensor data stored in InfluxDB through interactive dashboards. Time-series charts and gauge panels are created to display real-time and historical sensor values. Users can filter data by sensor type, location, and time range, providing a clear and intuitive overview of the hydroponic system's operational state.

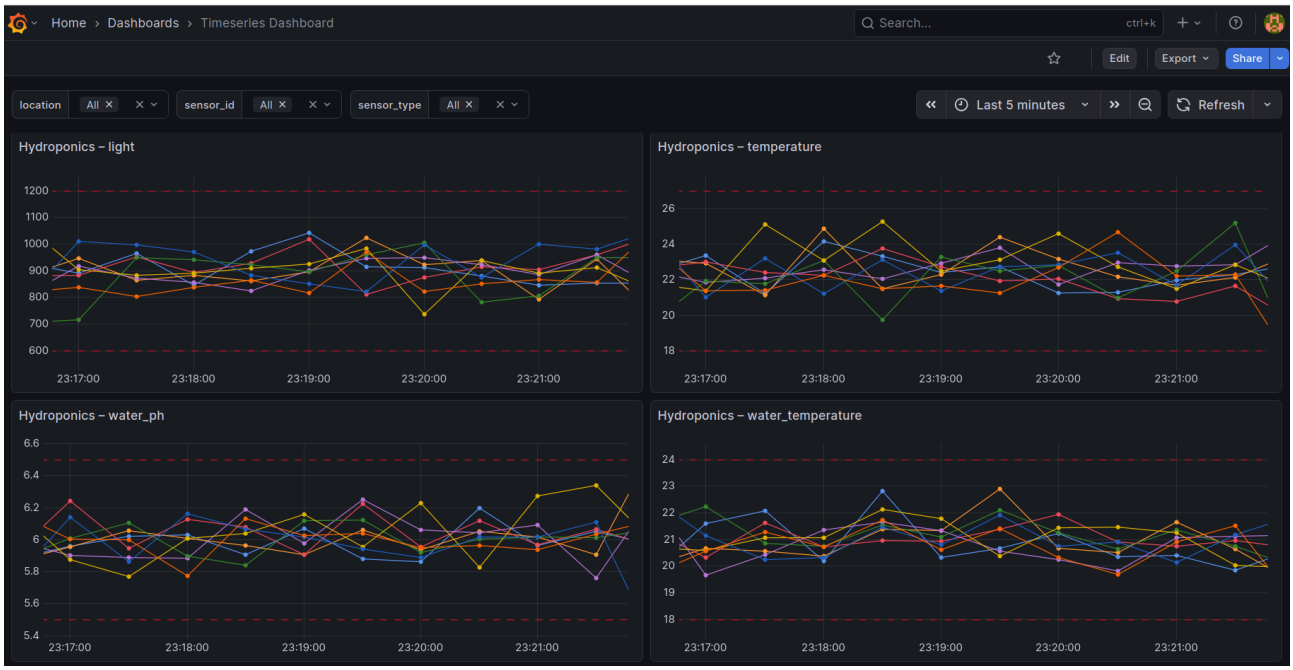

Figure 3: Time Series Data Visualization(Grafana)

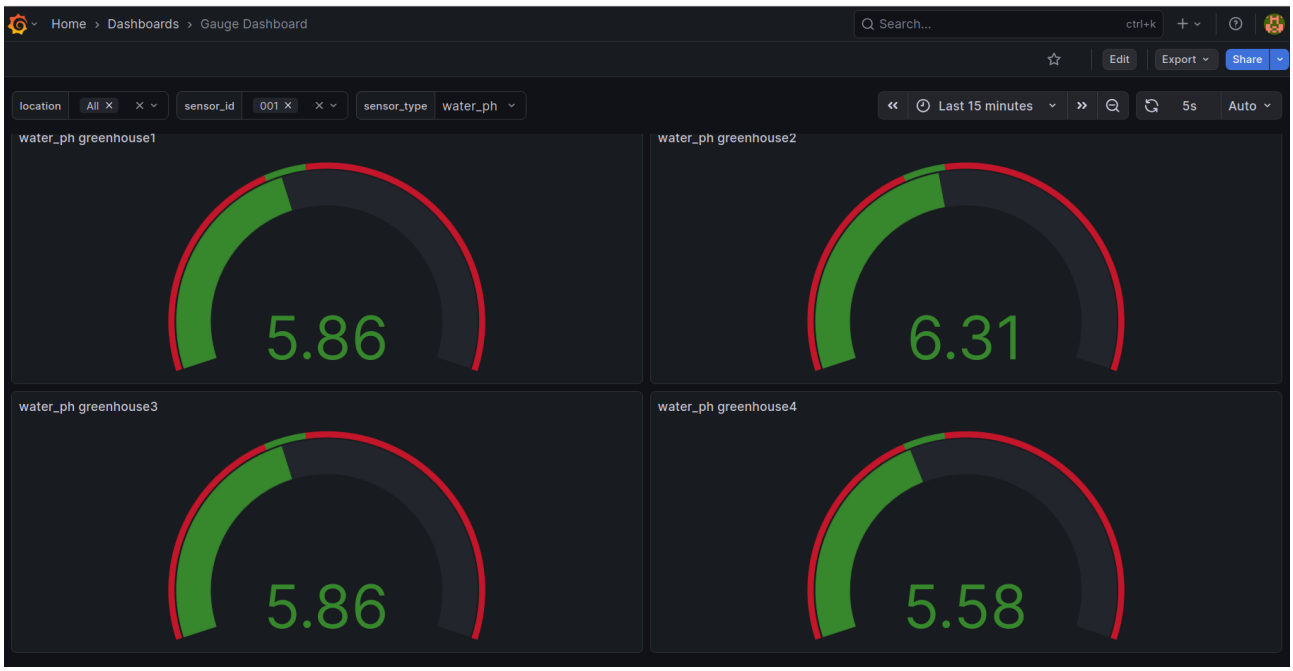

Figure 4: Gauge Dashboard(Grafana)

5.1.6 Telegram Bot (Notification and Alerting)

Telegram is used as the notification channel for delivering real-time alerts to users. A Telegram bot is integrated with Node-RED to send messages when sensor values violate predefined thresholds. Alert messages include information about the affected parameter, sensor location, and measured value, enabling users to respond promptly to critical system conditions.

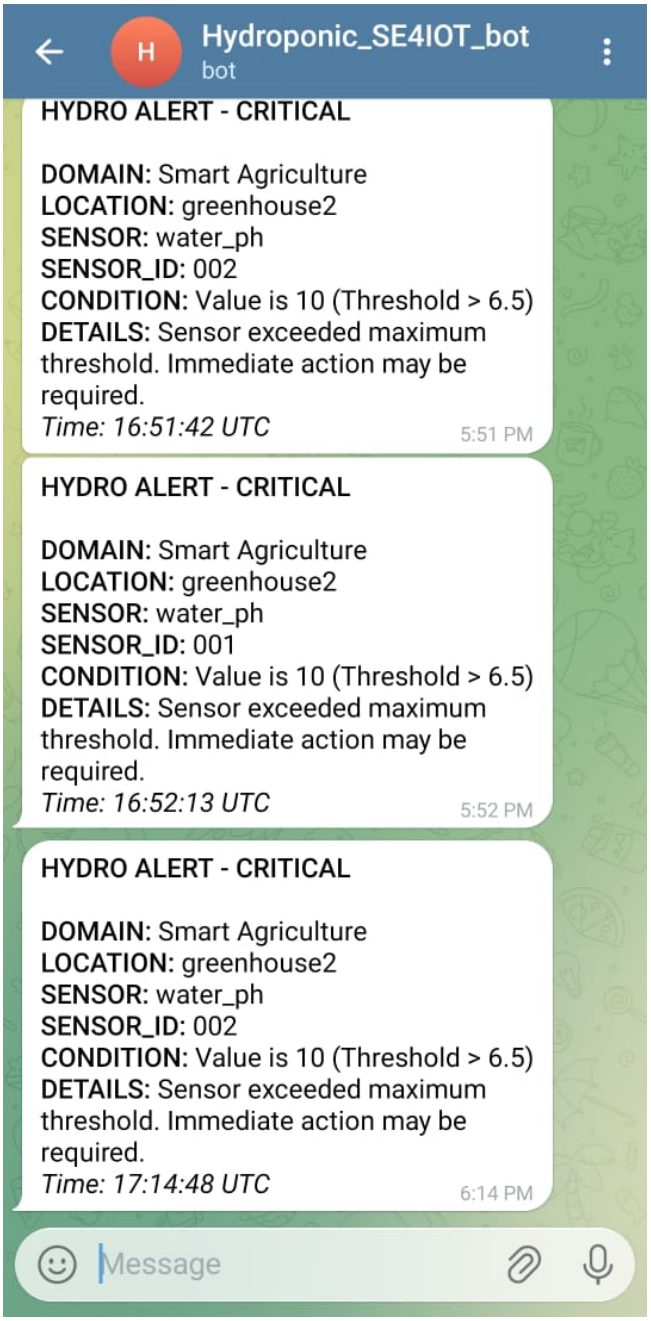

Figure 5: Telegram Bot Notification

5.2 Data Flow and Collaboration

The system operates as follows:

- The Python simulator generates sensor data continuously and publishes it to specific MQTT topics.
- The Mosquitto broker receives these messages and distributes them to all subscribed components.
- Node-RED consumes the sensor data, processes it, and evaluates it against defined thresholds.
- If a measurement is outside the safe range, Node-RED triggers a Telegram alert to notify the user.
- Simultaneously, sensor data is forwarded to InfluxDB, where it is stored for analysis and visualization.
- Grafana retrieves the data from InfluxDB and displays it on interactive dashboards, allowing users to monitor current and historical system conditions.

This collaboration between components ensures that the system operates autonomously: sensor data is continuously monitored, abnormal conditions are detected in real time, and users are immediately notified, all while storing historical data for analysis and decision-making.

5.3 Dynamic Configuration Management

This subsection describes the dynamic configuration framework implemented for managing sensor thresholds and system parameters across Node-RED, InfluxDB, and Grafana. The goal of this framework is to ensure that configuration changes - such as updates to `min`, `max`, `absolute_min`, and `absolute_max` values are applied automatically at runtime, without requiring container rebuilds, redeployments, or service restarts.

The solution enables a fully adaptive monitoring environment that supports rapid updates, consistent behavior across services, and zero operational downtime.

5.3.1 Motivation

No hard-coded values! And,
Typical Node-RED deployments load configuration data only once at startup. Any change in configuration files generally requires:

- redeploying the flow,
- restarting the Node-RED container,
- manual edits across multiple nodes.

This creates friction, increases maintenance effort, and risks inconsistencies between Node-RED, InfluxDB, and Grafana.
To address this, the system implements a *dynamic configuration loading mechanism* that continuously monitors a `config.yaml` file and automatically applies updates across the entire stack. This ensures:

- centralized configuration management,
- immediate application of new values,
- fully automated Grafana dashboard updates,
- no manual intervention required.

5.3.2 Dynamic Configuration Flow in Node-RED

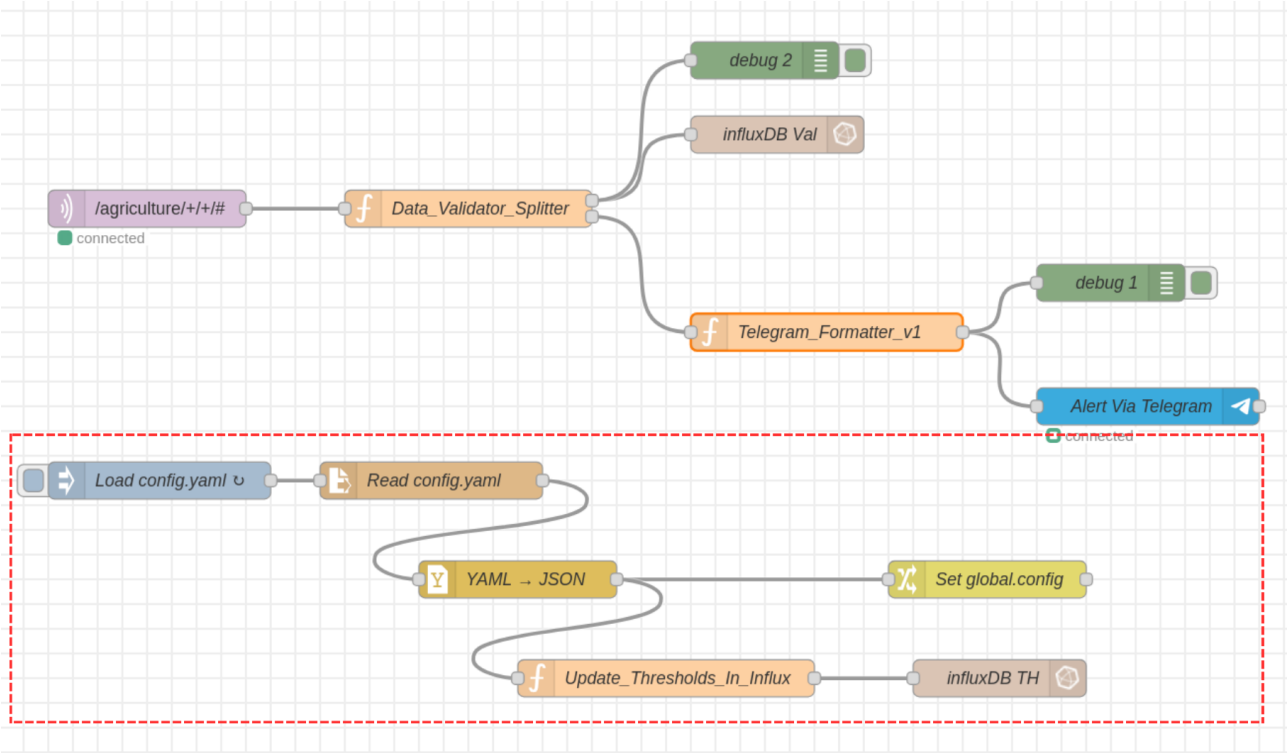

Figure 6: Configuration Flow

5.3.2.1 Overview Node-RED does not provide native hot-reloading of configuration files. To overcome this limitation, a dedicated flow was developed to continuously read and apply configuration updates from `config.yaml`.
The flow consists of:

1. a periodic trigger (every 60 seconds),
2. a file reader to load the YAML configuration,
3. a YAML-to-JSON conversion for Node-RED compatibility,

4. a global context update (`global.config`),
5. threshold synchronization with InfluxDB.

5.3.2.2 Flow Function In Figure 6, the diagriam highlighted in red demonstrates the following workflow:

- A scheduled trigger loads the `config.yml` file at fixed intervals.
- The YAML parser converts the configuration to JSON.
- The new configuration is compared to the currently stored version.
- If differences are detected, threshold values are updated inside InfluxDB.
- The global configuration is refreshed for real-time use by all flows.

5.3.2.3 Why This Flow Is Necessary This flow is essential because:

- Node-RED cannot detect file changes automatically,
- Node-RED does not re-import configuration data after startup,
- thresholds must remain external, centralized, and decoupled from flow logic,
- every component (Node-RED, InfluxDB, Grafana) must use the same authoritative configuration source.

Without this mechanism, configuration updates would require redeployments, interrupting operations and creating unnecessary maintenance overhead.

5.3.3 Threshold Management in InfluxDB

5.3.3.1 Purpose Threshold values such as:

- `min`,
- `max`,
- `absolute_min`,
- `absolute_max`

are treated as *system configuration*, not time-series sensor measurements. They change infrequently but are critical for alerting, visualization, and validation.

5.3.3.2 Storage Strategy Because these values are configuration metadata:

- they are stored in InfluxDB separately from sensor measurement data,
- only changes are written (to avoid unnecessary entries),
- each threshold is stored as a single latest-value record, ensuring Grafana always reads the newest configuration.

5.3.3.3 Benefits This approach provides:

- a centralized configuration store accessible to all services,
- fast lookup for Grafana panels,
- strict separation between operational configuration and historical sensor data,
- consistent, system-wide configuration integrity.

5.3.4 Grafana Dynamic Dashboard Updates

Grafana is configured to read threshold values directly from InfluxDB. This allows dashboard components (such as gauges, alert markers, and visualization bands) to update automatically whenever Node-RED writes new threshold values.

5.3.4.1 Outcomes The main outcomes are:

- no manual dashboard edits are required,
- no image rebuilds or redeployments are needed,
- visual ranges always match the current configuration,
- physical limits (`absolute_min`, `absolute_max`) define gauge boundaries,
- operational thresholds (`min`, `max`) define alert and warning bands.

Grafana becomes largely autonomous, adjusting its displays based on the latest live configuration.

5.4 Key Features Enabled by Architecture

- Autonomous Monitoring: Node-RED and Telegram provide automated evaluation and alerting without human intervention.
- Scalability: New sensors or locations can be added without modifying the core architecture, thanks to MQTT's topic-based communication.
- Resilience: The decoupled design ensures that the failure of one component does not halt the entire system.
- Real-Time Decision Making: Immediate data processing allows timely alerts and reduces the risk of crop damage.

6 Conclusion

The Smart Hydroponics IoT System demonstrates how autonomous software-driven solutions can enhance the efficiency, reliability, and responsiveness of modern hydroponic farming. By integrating sensor simulation, MQTT-based communication, real-time data processing through Node-RED, time-series storage in InfluxDB, visualization with Grafana, and instant alerts via Telegram, the system provides continuous monitoring of critical environmental parameters. This project highlights key principles of autonomous systems, including real-time decision-making, threshold-based automated responses, scalability, and modular design. The fully containerized deployment ensures portability and ease of maintenance, while the interactive dashboards and alerting mechanisms enable proactive management of hydroponic environments.

Overall, the system serves as a practical example of applying IoT and software engineering concepts to create a reliable, autonomous monitoring solution for smart agriculture. Future enhancements could include integration with physical hydroponic hardware, predictive analytics for crop optimization, and adaptive control strategies to further increase system autonomy and efficiency.

Project Repository

The full source code, configuration files, and Docker setup for this project are publicly available at: https://github.com/tesfawe/se4iot_hydroponics_project