Revista Brasileira DE ESTUDOS CIENTÍFICOS

Implementation and Case Study of a Modular IoT Monitoring Architecture

Luiz Fernando M. Arruda
luiz.arruda@ieee.org
0000-0002-9159-2245
Centro Universitário Avantis - UNIAVAN

João Airton G. de Sousa
joao.airton@uniavan.edu.br
0009-0003-5121-0956
Centro Universitário Avantis - UNIAVAN

***Abstract* – This manuscript presents a practical implementation and integration case study of a modular Internet of Things (IoT) monitoring architecture based on widely adopted open-source technologies. The system is structured into four functional layers, Broker, Processing, Storage, and Visualization, organizing the end-to-end data flow from sensor acquisition to dashboard representation. The architecture was implemented using an ESP32 device with environmental sensors, MQTT communication, Node-RED middleware, InfluxDB Cloud for time-series storage, and Grafana for visualization. The objective of this study is not to propose a novel architectural paradigm, but to provide a clear and reproducible reference implementation that illustrates how these components can be coherently integrated into a modular workflow. The experimental deployment confirms the functional consistency of the layered organization and demonstrates its applicability in small-scale IoT monitoring scenarios.**

***Keywords* – IoT, modular architecture, MQTT, Node-RED, InfluxDB, Grafana**

Recebido em: 19/03/2026
Revisado em: 17/06/2026
Aprovado em: 01/07/2026

1 INTRODUCTION

The Internet of Things (IoT) has expanded significantly in recent years, supporting applications that range from smart homes and industrial automation to healthcare and environmental monitoring (Fig. 1) [1]. In general, IoT devices combine sensors, actuators, and communication modules, enabling them to collect data from their surroundings and exchange information through networked systems [2].

Microcontrollers such as the ESP32 have become popular in IoT prototypes due to their low cost, built-in connectivity, and ease of use, attributes that make them suitable for small-scale deployments and rapid development [3]. Despite the availability of versatile hardware, the practical integration of IoT systems into coherent and maintainable architectures remains a common challenge. Devices are frequently assembled in ad hoc configurations, which may complicate interoperability and long-term maintainability [4].

Some works have attempted to mitigate these limitations. For instance, [5] proposed a real-time monitoring system that integrates environmental data from classrooms, study rooms, and other campus facilities. Their approach demonstrates how heterogeneous IoT devices can be orchestrated to support both operational decision-making and analytical tasks in large-scale educational environments.

Fig. 1. Common IoT devices used in various applications.

Beyond the challenges of integration, IoT deployments rely on diverse communication protocols. As noted in [6], supporting multiple protocols, such as MQTT, HTTP, and TCP, requires balancing bandwidth, latency, and reliability constraints. MQTT, in particular, has gained relevance because of its lightweight design, low overhead, built-in publish/subscribe model, and suitability for intermittent or resource-constrained devices.

Complementary technologies have also matured around these communication protocols. MQTT brokers facilitate message routing among heterogeneous clients; tools like Node-RED simplify the creation of processing flows; and databases such as MongoDB, NoSQL solutions, and time-series platforms like InfluxDB provide structured mechanisms for storing high-frequency data. Visualization tools, including Grafana and Kibana, help close the loop by offering dashboards that support real-time analysis of the collected information [7].

Layered architectures are widely adopted in IoT systems to separate concerns related to data acquisition, communication, processing, and application services. Rather than proposing a new architectural paradigm, this work presents a practical implementation and integration case study of a modular IoT monitoring architecture structured into four functional layers: Broker, Processing, Storage, and Visualization. The objective is to demonstrate how widely adopted open-source technologies can be arranged into a coherent workflow that supports transparent data flow from sensor acquisition to dashboard visualization. The focus of this study is on architectural clarity, interoperability between layers, and reproducibility of the integration process, rather than on large-scale performance benchmarking or formal

DOI: 10.5281/zenodo.21106435

architectural innovation.

The remainder of this paper is structured as follows. Section 2 introduces the modular IoT framework; Section 3 describes its implementation in a practical test scenario; and Section 4 summarizes the main findings and outlines directions for future evaluation.

2 MODULAR ARCHITECTURE

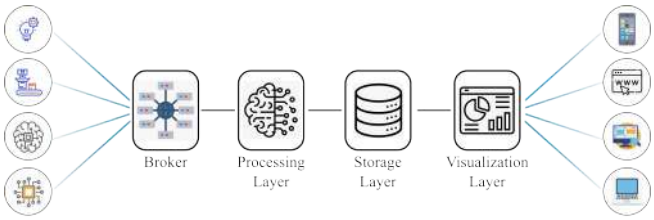

Fig. 2. A Modular Architecture for IoT Devices.

Fig. 2 illustrates the modular architecture for IoT monitoring and data processing. The architecture is composed of four main layers: (i) Broker Layer, (ii) Processing Layer, (iii) Storage Layer, and (iv) Visualization Layer. Each layer is responsible for specific functionalities within the IoT ecosystem, ensuring efficient data flow and management from device communication to data visualization.

2.1 BROKER

The Broker Layer manages the communication between IoT devices and the central system, acting as the first point where data produced by sensors and actuators is received and organized. In practice, this layer handles the ingestion of messages from different devices and ensures that the information is forwarded to the appropriate destinations. Typical components include message brokers, parsers, and protocol translators. Because IoT deployments often rely on multiple communication standards, the broker must support protocols such as MQTT, CoAP, and HTTP to accommodate devices with different capabilities and communication requirements [8].

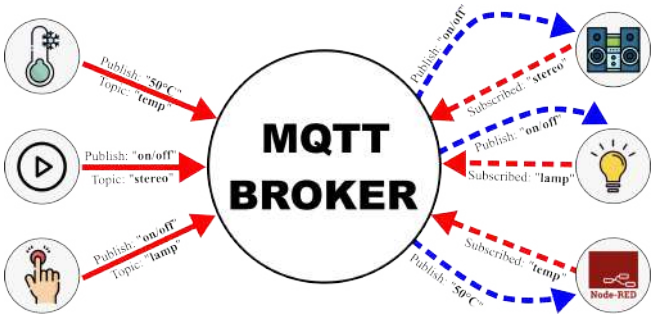

Fig. 3. Broker Layer.

Fig. 3 illustrates an example of how this layer is typically organized. On the left, various IoT devices publish data according to their own communication constraints. At the center of the diagram, the broker receives these messages and arranges them into hierarchical topics, which helps structure the information and maintain an organized flow. On the right, several clients act as subscribers and receive only the topics they are interested in, allowing different applications or services to consume the same data stream independently.

Beyond routing messages, modern brokers incorporate mechanisms that help maintain reliable communication even under changing network conditions. These include managing the sessions of devices that may temporarily disconnect, retaining messages so late subscribers can access the most recent values, and routing topics hierarchically to simplify data organization. Brokers also use persistent queues and buffering strategies to avoid data loss during periods of instability. In distributed or clustered configurations, they may synchronize session states and replicate topics across nodes, improving service continuity and overall system resilience [9].

2.2 PROCESSING LAYER

The Processing Layer is responsible for handling the data received from the Broker Layer and preparing it for storage. At this stage, the system must filter, normalize, and reorganize the information so that it can be used consistently by downstream components. To support these tasks, a middleware solution is adopted to coordinate the processing flows and manage how data moves through the architecture.

The middleware offers a modular environment in which workflows are built by connecting functional blocks, each one performing a specific operation such as receiving MQTT messages, checking the validity of the payload, transforming formats, or forwarding results to other services. This approach provides a practical way to integrate IoT devices, web services, and heterogeneous data sources within the same framework [10].

In the implementation used in this work, the middleware subscribes to the topics managed by the broker and receives the incoming messages directly from the sensor node. The processing blocks interpret the payloads, extract the relevant fields, discard inconsistent readings, and reorganize the data into a structured format, commonly a JSON object. This pre-processing step ensures that the information is consistent with the requirements of the system and can be transferred without friction to the storage layer, including time-series databases that rely on clean and well-defined structures [11].

Because it operates between the device layer and the storage solutions, the Processing Layer functions as an integration hub. It is here that the logic governing data handling is effectively implemented through the connections between input, transformation, and output components [13]. This positioning allows the middleware to absorb differences in protocols and formats, providing a single point where data can be translated, normalized, or enriched before moving forward. As a result, applications that consume the processed data are not tied to device-specific constraints or communication details [14].

Another advantage of adopting modular processing blocks

DOI: 10.5281/zenodo.21106435

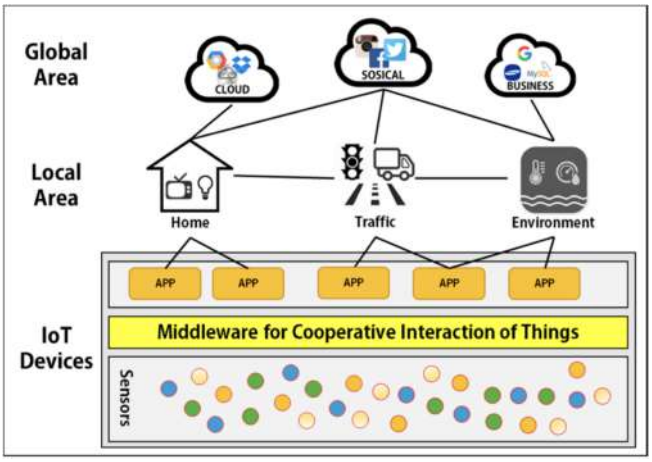

Fig. 4. IoT overview based on middleware [12].

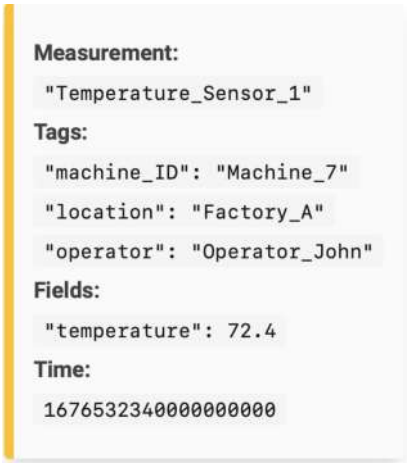

Fig. 5. Time-series databases for IoT [18].

is maintainability. Individual functions can be modified or extended without disrupting the remaining workflow, which makes it easier to include new filtering operations, add additional services, or alter the behavior of existing components. This contributes to the scalability of the system, allowing the architecture to evolve naturally as new requirements emerge [12], [15].

Fig. 4 shows an example of a multi-layer IoT architecture in which heterogeneous sensors send their data to a middleware layer. This layer coordinates the interaction among applications that support various services, such as home automation, traffic management, and environmental monitoring.

2.3 STORAGE LAYER

The Storage Layer is responsible for keeping the processed data organized and accessible so that it can be used reliably by the rest of the system. In IoT applications, sensor readings typically form time-series data, which demands storage solutions capable of handling frequent writes, applying retention rules, and supporting time-based queries. For this reason, databases designed specifically for time-series information, as well as scalable NoSQL systems, are commonly adopted in these scenarios [16].

In the architecture used here, the Storage Layer follows a hierarchical structure that helps organize the data according to its characteristics and expected retention period. At the top level, containers group datasets that share similar storage rules. Inside each container, the data is arranged in measurement structures, conceptually similar to tables in relational databases, that bring together readings of the same type, such as temperature or humidity [17].

Each data point stored in the system includes a timestamp and two categories of information: tags and fields. Tags are indexed key–value pairs that describe metadata such as the originating device or its location, while fields store the measurement values themselves and are not indexed. This separation, illustrated in Fig. 5, allows the database to filter data efficiently based on metadata while still supporting the storage of large volumes of continuous sensor readings [19]. Through this structure, the Storage Layer becomes the central repository for all recorded information and provides the foundation on which visualization tools can explore real-time behavior or analyze long-term trends.

Another important role of this layer is to ensure that data is preserved even when the system experiences unstable conditions. IoT deployments often face intermittent connectivity, device failures, or changes in data rates. To maintain consistency, the storage system relies on mechanisms such as write-ahead logging, replication across multiple nodes, and periodic snapshots, which help prevent data loss and keep the system operational under adverse circumstances [20].

Besides supporting durability, the Storage Layer also enables efficient analytical processing. By storing data in a time-structured format enriched with metadata, it allows upper layers to perform tasks such as aggregation, windowing, anomaly detection, and pattern analysis with minimal overhead. This organization benefits both real-time and batch processing and supports the integration of statistical models and machine learning techniques. Consequently, the Storage Layer serves not only as a repository but as a key component for generating insights and guiding data-driven decisions throughout the system [21], [22].

2.4 VISUALIZATION LAYER

The Visualization Layer provides the tools through which users can examine and interact with the information stored in the system. Its purpose is to convert processed data into interpretable views such as dashboards, analytical panels, and real-time displays that help users follow the behavior of the monitored environment [23]. Fig. 6 illustrates an example of visualization resources implemented using Grafana.

In the architecture, this layer constitutes the final stage of the data path. Once the information has been collected, processed, and stored, visualization tools transform time-series metrics into actionable insights through elements such as line charts, gauges, and summary tables. Because the data is retrieved directly from the Storage Layer, the dashboards

DOI: 10.5281/zenodo.21106435

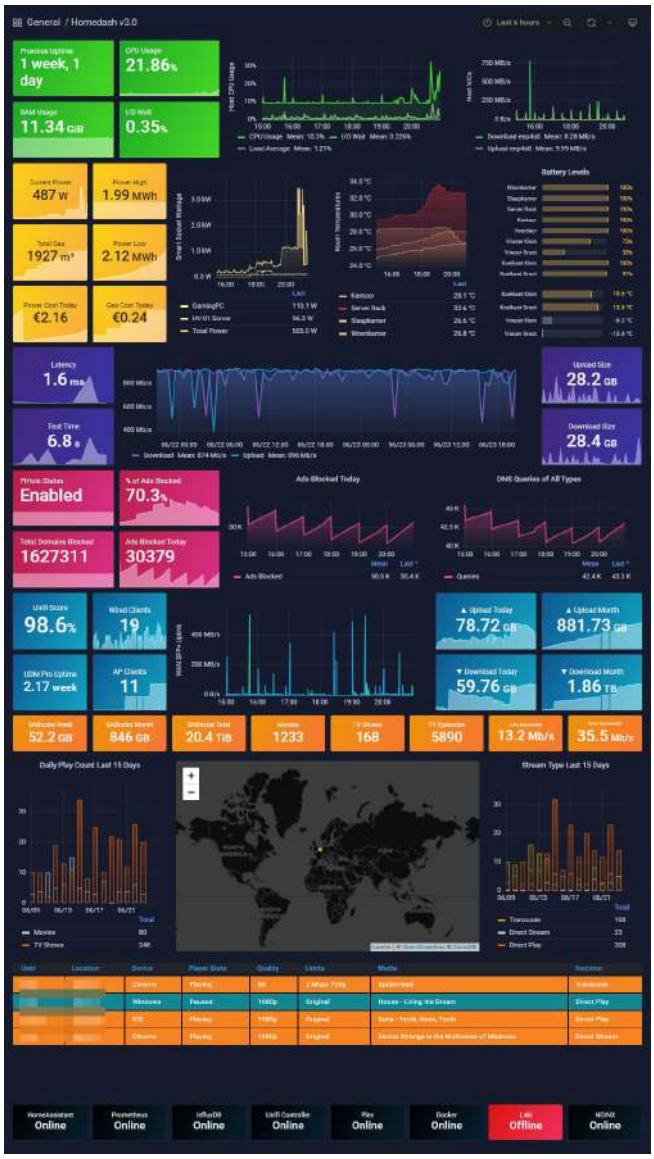

Fig. 6. Sample of resources in the Visualization Layer using Grafana [24].

reflect both real-time system activity and long-term trends without requiring modifications to earlier processing stages.

The importance of visualization in IoT systems extends well beyond displaying raw measurements. According to [23], it is a central component of the visual analytics pipeline, enabling the exploration of heterogeneous and high-volume data through interactive representations that reveal patterns, correlations, and anomalies that may not be evident otherwise. Their survey highlights that effective dashboards should support real-time updates, contextual filtering, and multiple visualization modes tailored to the characteristics of the underlying data. This aligns with the design adopted in our architecture, where Grafana dashboards provide both operational awareness and historical analysis of sensor readings.

Challenges become more pronounced in large IoT deployments, where thousands of data streams may exhibit highly diverse rates and behaviors. As discussed by [25], traditional broker consoles and generic dashboards often fail to provide meaningful situational awareness in such settings. Their work demonstrates that effective visualization at scale requires real-time responsiveness, aggregation of hierarchical streams, and inspection tools capable of exposing connectivity issues, latency spikes, or malformed payloads “at a glance.” These requirements reinforce the relevance of adopting a platform capable of correlating environmental measurements with communication indicators, as implemented in this work through Grafana panels that simultaneously display sensor data and Wi-Fi signal strength.

Visualization also plays a critical role in industrial and large-scale IoT environments. [26] emphasize that dashboards must support different user profiles,ranging from technicians monitoring equipment conditions to managers assessing operational KPIs, while providing responsive and adaptable interfaces. Their platform illustrates how Grafana can consolidate operational metrics, energy consumption, predictive analytics, and anomaly detection results into multi-tier dashboards tailored to distinct decision-making needs. This perspective strengthens the motivation for employing flexible visualization tools capable of evolving alongside system requirements.

Common dashboard features such as time-range selection, metadata-based filters, and automatic updates help users explore the data from multiple viewpoints, identify irregularities, and compare variables under different conditions. These interactions assist in detecting performance deviations or contextual anomalies that are not immediately visible from raw time-series values.

By closing the loop between acquisition and interpretation, the Visualization Layer enables informed assessments of the monitored environment. The ability to share dashboards among multiple users also encourages collaborative analysis, allowing teams to annotate findings, compare observations, and coordinate responses in operational and supervisory contexts.

Finally, because the visualization tools operate independently of the lower layers, new graphical components or domain-specific widgets can be introduced without altering the underlying architecture. As noted in [27], real-time visualization is essential for detecting deviations promptly, and the integration of Grafana with time-series databases such as InfluxDB enhances the capacity to manage and analyze large volumes of IoT data efficiently. This combination supports interactive dashboards capable of revealing trends and anomalies across different time scales [28].

2.5 INTEGRATION

The architecture is organized into a set of layers that operate together to sustain the full data flow of an IoT system, as shown in Fig. 7. Although each layer has a specific role, they remain loosely coupled so that changes in one component do not require adjustments in the others. This structure helps maintain clarity in the design and supports the scalability and

DOI: 10.5281/zenodo.21106435

flexibility needed for IoT deployments.

Fig. 7. Layer Integration in IoT Architecture.

The process begins in the Broker Layer, which acts as the central communication point. Devices publish their measurements to predefined topics, and the broker distributes these messages to any component that has subscribed to them. This publish/subscribe model separates producers from consumers and makes it easier for multiple services to access the same data stream without direct connections.

Once the data reaches the Processing Layer, it is interpreted and reorganized. The middleware subscribed to the broker topics applies operations such as filtering, format conversion, validation, and the inclusion of contextual metadata. Because these tasks are implemented through modular components, the processing workflow can be adapted or expanded without affecting how devices communicate or how data is stored.

After processing, the information is forwarded to the Storage Layer, where it is kept in a structured form that supports time-based queries and long-term retention. This layer handles the durability mechanisms and indexing strategies required to preserve data reliably, serving as the repository used by analytical tools and monitoring services.

The final stage occurs in the Visualization Layer, which retrieves the stored measurements and displays them through dashboards or analytical panels. Since the visualization tools interact directly with the data repository, users can explore both recent activity and historical patterns without needing to adjust the underlying processing logic. Configurable elements such as time selectors and filters help adapt the visual output to different needs.

Altogether, the four layers form a coherent workflow in which data moves from acquisition to interpretation in a structured way. The separation of responsibilities simplifies maintenance and allows different technologies to be integrated without compromising the overall system. This organization supports an architecture that can be expanded, reorganized, or adapted as new requirements emerge.

3 METHODOLOGY AND APPLICATION

The modular architecture was evaluated in practice through an environmental monitoring experiment that measured temperature and humidity. This evaluation validated the end-to-end data flow and the functional integration of the proposed layers, demonstrating how the components interact coherently and how data is reliably propagated throughout the system. The focus of the evaluation was on architectural consistency and interoperability rather than on quantitative performance benchmarking.

3.1 EVALUATION SCHEME

The system was built around an ESP32 connected to the local Wi-Fi network and equipped with a DHT11 temperature and humidity sensor (Fig. 8). With this setup, the device periodically sent measurements to a locally installed MQTT broker, enabling continuous monitoring of the environment.

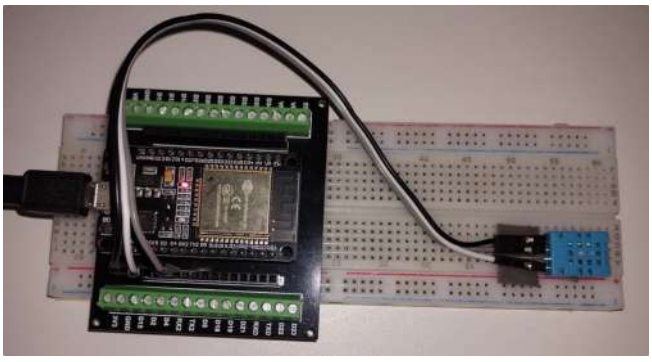

Fig. 8. Evaluation scheme used in the environmental monitoring test.

3.2 BROKER LAYER

The Broker Layer receives the sensor readings published by the ESP32 and forwards them to the remaining components of the architecture. Communication is managed through an

MQTT broker hosted on HiveMQ Cloud, which serves as the system's entry point by organizing and routing incoming messages. In the prototype, the ESP32 equipped with a DHT11 sensor connects to the local Wi-Fi network and periodically sends temperature and humidity values to the topic "casa/quarto/dht_clima". The connection parameters used in the application: server address, user credentials, port, and topic configuration, are shown in Fig. 9. This setup provides a lightweight and reliable mechanism for validating the architecture's end-to-end data flow.

```
// ---------- MQTT (HiveMQ Cloud) ----------
const char* MQTT_SERVIDOR = "4ac678f8c99146b183909ee56966d767.s1.eu.hivemq.cloud";
const int   MQTT_PORTA    = 8883;
const char* MQTT_USUARIO  = "*****";
const char* MQTT_SENHA    = "***************";

// Tópico do protótipo
const char* MQTT_TOPICO = "casa/quarto/dht_clima";
```

Fig. 9. HiveMQ Cloud configuration used in the Broker Layer.

3.3 PROCESSING LAYER

In the Processing Layer, the middleware was implemented using Node-RED (Fig. 10). In this configuration, Node-RED subscribed to the MQTT topic published by the ESP32 and handled the messages as they arrived. The first node in the flow was the MQTT input block, which forwarded each message to a function node responsible for structuring the payload.

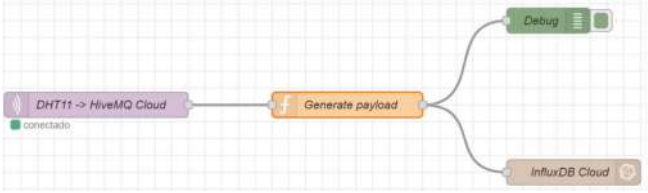

Fig. 10. Processing Layer flow implemented in Node-RED.

At this stage, the measurements were organized into a JSON object containing temperature, humidity, and the Wi-Fi signal strength. Additional metadata, such as the measurement name used in the database and the timestamp, was also inserted. For debugging purposes, the messages could be routed in parallel to a debug node, which helped verify the data during implementation.
The processing flow performed the following tasks:

- filtering and validating incoming messages;
- converting the raw payload into a structured JSON format;
- adding metadata such as device identifier and location;
- forwarding the normalized data to the database.

An example of a structured payload generated by the middleware is shown below:

```
{
  "topic": "casa/quarto/dht_clima",
  "package": {
    "temperature": 25,
    "humidity": 71,
    "wifi_rssi": -51
  },
  "qos": "0",
  "retain": false,
  "_msgid": "bf9ce88ebeeea5d3",
  "measurement": "room_environment",
  "timestamp": 17643116943
}
```

3.4 STORAGE LAYER

After processing, the data was sent to the *Storage Layer*, implemented using InfluxDB Cloud. Each record was inserted into the *room_environment* measurement, which stored the temperature, humidity, and Wi-Fi signal strength fields together with the timestamp created in Node-RED. Fig. 11 shows a portion of the stored time series obtained through an SQL query, illustrating the continuity of the readings over time.

Fig. 11. Stored environmental data in InfluxDB Cloud.

3.5 VISUALIZATION LAYER

In the Visualization Layer, the data stored in InfluxDB was accessed directly by a Grafana dashboard. This dashboard displayed the evolution of temperature and humidity using time-series graphs, allowing the behavior of the monitored environment to be observed in real time. Figure 12 shows the configured panel.
A separate panel was included to display the *Wi-Fi Signal Strength*, which made it possible to relate signal variations to the continuity of the received measurements.

Fig. 12. Visualization Layer with Grafana dashboard.

3.6 LIMITATIONS

Although the proposed architecture was validated through a functional deployment, this study presents certain

DOI: 10.5281/zenodo.21106435

limitations. First, the evaluation was conducted in a small-scale experimental environment involving a single ESP32 node and a limited number of data streams. Consequently, the results do not characterize system behavior under high-load or large-scale deployment conditions.

Second, no formal quantitative benchmarking was performed to measure metrics such as end-to-end latency, throughput under stress, or broker performance under varying publication rates. The validation focused primarily on verifying architectural coherence and interoperability between layers.

Third, security aspects such as authentication mechanisms, encryption policies, and resilience against malicious traffic were not explicitly evaluated. While the adopted technologies provide built-in security capabilities, their configuration and performance impact were outside the scope of this work.

Finally, the system relies partially on cloud-based services, which may introduce dependencies related to network availability and service-level agreements. Future investigations may extend this study by incorporating stress testing, security analysis, and comparative benchmarking across alternative architectural configurations.

4 CONCLUSION

This work presented a practical implementation of a modular IoT monitoring architecture organized into four layers: Broker, Processing, Storage, and Visualization. The prototype, based on an ESP32 and supported by Node-RED, InfluxDB Cloud, and Grafana, enabled observation of component interaction and end-to-end data flow from acquisition to visualization.

The experimental deployment demonstrated that the layered organization supports coherent interaction throughout the data flow. The broker ensured reliable communication, the middleware structured and enriched incoming messages, the storage layer preserved time-series data consistently, and the visualization tools provided clear monitoring insights. Together, these elements show that the modular organization facilitates the integration of heterogeneous devices in small-scale IoT scenarios.

Future extensions may include quantitative evaluation under higher publication rates, analysis of communication latency, and assessment of security configurations in more demanding deployment scenarios.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[3] P. Foltynek, M. Babiuch, and P. Suranek, "Measurement and data processing from internet of things modules by dual-core application using esp32 board," *Measurement and Control*, vol. 52, no. 7-8, pp. 970–984, 2019.

[4] T. Dominguez-Bolano, V. Barral, C. J. Escudero, and J. A. Garcia-Naya, "An iot system for a smart campus: Challenges and solutions illustrated over several real-world use cases," *Internet of Things*, vol. 25, p. 101099, 2024.

[5] P. Macheso, T. D. Manda, S. Chisale, N. Dzupire, J. Mlatho, and D. Mukanyiligira, "Design of esp8266 smart home using mqtt and node-red," in *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, 2021, pp. 502–505.

[6] L. Thomas, M. K. MV, S. D. SL, and P. BS, "Towards comprehensive home automation: Leveraging the iot, node-red, and wireless sensor networks for enhanced control and connectivity," *Engineering Proceedings*, vol. 59, no. 1, 2023.

[7] A. Prakash and T. Vengattaraman, "IoT-based Smart University Using MQTT Protocol: A Scalable and Sustainable Design Approach," in *2024 5th International Conference on Communication, Computing & Industry 6.0 (C2I6)*, 2024, pp. 1–7.

[8] E. Longo, A. E. C. Redondi, M. Cesana, and P. Manzoni, "Border: A benchmarking framework for distributed mqtt brokers," *IEEE Internet of Things Journal*, vol. 9, no. 18, pp. 17 728–17 740, 2022.

[9] O. Standard, "Mqtt version 3.1. 1," *URL http://docs. oasis-open. org/mqtt/mqtt/v3*, vol. 1, p. 29, 2014.

[10] M. Turilli, V. Balasubramanian, A. Merzky, I. Paraskevakos, and S. Jha, " Middleware Building Blocks for Workflow Systems ," *Computing in Science & Engineering*, vol. 21, no. 04, pp. 62–75, Jul. 2019.

[11] T. Joseph, R. Jenu, A. K. Assis, V. A. S. Kumar, P. M. Sasi, and G. Alexander, "Iot middleware for smart city: (an integrated and centrally managed iot middleware for smart city)," in *2017 IEEE Region 10 Symposium (TENSYMP)*, 2017, pp. 1–5.

[12] S. Jeon and I. Jung, "Mint: Middleware for cooperative interaction of things," *Sensors*, vol. 17, no. 6, 2017.

[13] F. Z. Amara, M. Hemam, M. Djezzar, and M. Maimor, "Semantic web and internet of things: Challenges, applications and perspectives," *Journal of ICT Standardization*, vol. 10, no. 2, pp. 261–291, 2022.

[14] G. Fersi, "Middleware for internet of things: A study," in *2015 International Conference on Distributed Computing in Sensor Systems*, 2015, pp. 230–235.

[15] L. D. Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[16] S. Reetishwaree and V. Hurbungs, "Evaluating the performance of sql and nosql databases in an iot environment," in *2020 3rd International Conference on Emerging Trends in Electrical, Electronic and Communications Engineering (ELECOM)*, 2020, pp. 229–234.

[17] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013, including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond.

[18] HiveMQ. (2023) Iiot data storage and actionable analytics generation. Accessed: 2025-11-25. [Online]. Available: https://www.hivemq.com/blog/iiot-data-storage-and-actionable-analytics-generation/

[19] J. Han, H. E, G. Le, and J. Du, "Survey on nosql database," in *2011 6th International Conference on Pervasive Computing and Applications*, 2011, pp. 363–366.

[20] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–10.

[21] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018.

[22] M. ABU KAUSAR and M. Nasar, "Suitability of influxdb database for iot applications," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 10, pp. 10–35 940, 2019.

[23] A. Protopsaltis, P. Sarigiannidis, D. Margounakis, and A. Lytos, "Data visualization in internet of things: tools, methodologies, and challenges," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ser. ARES '20. New York, NY, USA: Association for Computing Machinery, 2020.

[24] Grafana Community. (2020) Removing graph title bars and moving them inside. Accessed: 2025-11-25. [Online]. Available: https://community.grafana.com/t/removing-graph-title-bars-and-moving-them-inside/107708

[25] A. Taivalsaari and T. Mikkonen, "Visualizing streaming data in industrial scale iot systems," in *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*, ser. SAC '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 663–670.

[26] S. Kahveci, B. Alkan, M. H. Ahmad, B. Ahmad, and R. Harrison, "An end-to-end big data analytics platform for iot-enabled smart factories: A case study of battery module assembly system for electric vehicles," *Journal of Manufacturing Systems*, vol. 63, pp. 214–223, 2022.

[27] A. Mehdi, M. K. Bali, S. I. Abbas, and M. Singh, ""unleashing the potential of grafana: A comprehensive study on real-time monitoring and visualization"," in *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, 2023, pp. 1–8.

[28] A. Aggoune and Z. Benratem, "Ecg data visualization: Combining the power of grafana and influxdb," in *2023 International Conference on Advances in Electronics, Control and Communication Systems (ICAECCS)*, 2023, pp. 1–6.