Performance and Resilience Evaluation of a Hybrid Edge–Cloud IoT System Across Connected, Offline, and Reconnection Scenarios

Luís Fernando Alves da Silva(✉), Pedro Ramires da Silva Amalfi Costa, and Claudio Schepke

Laboratory of Advanced Studies in Computation, Federal University of Pampa (UNIPAMPA), Alegrete, RS, Brazil
{luisfads.aluno,pedroamalfi.aluno,claudioschepke}@unipampa.edu.br

Abstract. Reliable operation in Internet of Things (IoT) systems is crucial in environments with intermittent connectivity and unstable infrastructure, such as rural agricultural settings. When using cloud-centric architectures that rely on continuous internet access, users often face issues such as service disruptions, data loss, and a lack of real-time control during connectivity degradation. To overcome these limitations, this paper proposes a hybrid edge-to-cloud architecture that enables autonomous operation, resilient device control, and consistent data synchronization under unreliable network conditions. The proposed architecture integrates local edge processing, event-driven replication, and asynchronous message-based synchronization using RabbitMQ, Celery, and MQTT. It enables IoT devices to continue functioning in offline mode, maintain low-latency execution of control logic, and automatically reconcile the application state once connectivity returns. To assess system performance under realistic stress conditions, we conducted a load-testing experiment evaluating the behavior of the edge node across three connectivity states: fully connected to the cloud, operating autonomously during disconnection, and during the reconnection phase with backlog synchronization. Scripted message publishers emulating IoT devices produce workloads of increasing intensity. That enables quantitative analysis of throughput, end-to-end latency, resource consumption, and backlog draining time across progressively escalating load levels. These findings suggest that hybrid Edge–Cloud IoT architectures based on open, interoperable technologies offer a sustainable path toward technological independence, thereby reducing reliance on proprietary cloud platforms and enabling resilient real-time operation in critical domains where autonomy and fault tolerance are essential.

Keywords: Edge Computing · Hybrid Edge–Cloud Architecture · IoT Systems · Intermittent Connectivity · Offline Operation · Event-Driven Synchronization · Load Testing · Performance Evaluation

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026, LNCS 16768, pp. 651–668, 2027.
https://doi.org/10.1007/978-3-032-30488-9_40

1 Introduction

Agriculture is essential for human subsistence. Technological advances have enabled the automation of various tasks, consolidating the concept of Agriculture 4.0, in which the integration of sensors, connectivity, and computational intelligence allows for the optimization of production processes [1]. Irrigation, a critical element in agricultural production, can be improved by smart technologies, ensuring more efficient use of water and other resources [3].

Implementing systems in rural areas presents specific challenges. The lack of reliable internet infrastructure and adverse weather conditions can compromise irrigation continuity. Thus, system security is fundamental to ensuring its robust operation, especially regarding its IoT architecture.

Given this, we identified the need for automated irrigation systems to address the inherent challenges of irrigation automation on small rural properties. That includes recurring losses due to power outages and water supply interruptions. The developed system aims to mitigate frequent problems that negatively impact agricultural production. The proposed system monitors environmental data and controls irrigation flow using an IoT infrastructure with a hybrid cloud-edge approach. It allows for disconnected operation in case of connection failures. Furthermore, the system design intends to be accessible to users without technological familiarity, ensuring ease of use.

The contribution of this paper is to provide an economically viable IoT irrigation system for small rural producers. Research conducted with small rural producers at the regional level during the development of this solution identified that several producers are interested in smart irrigation solutions. However, most of them respond that the high price of these systems is the main impediment to their purchase and use. This system aims to address a market niche that is currently underexplored by smart irrigation systems, as they typically focus on automating pivots used by large rural producers.

The organization of the remainder of the paper is as follows. Section 2 presents the related work. Section 3 describes the methodology of the experiments. Next, in Sect. 4, we describe the process of implementing the components of the edge-cloud application. Section 5 shows the experimental results. At last, Sect. 6 presents the conclusion and future work.

2 Related Work

Cloud-Centric platforms enable the collection, storage, and analysis of data on a large scale, offering consolidated tools for visualization and integration. The Smart Water Management Platform (SWAMP) is an example of an architecture that adopts FIWARE, an open ecosystem funded by the European Union, composed of software modules called Generic Enablers [6]. These modules provide essential functionalities for IoT applications, such as context management, sensor and actuator integration, historical data storage, and large-scale processing. In SWAMP, FIWARE serves as an intermediary layer between field devices

and analytical applications, ensuring interoperability, scalability, and support for real-time decision-making in precision irrigation. Furthermore, it integrates IoT protocols and persistence components to support operations in real-world scenarios while highlighting performance and scalability challenges in the context of database modules. Although cloud centralization improves aspects such as governance and elasticity, the strong dependence on stable connectivity represents a significant limitation in rural areas, subjecting them to network intermittency, a typical condition faced by small producers.

Hybrid architectures intend to balance cloud processing and storage resources with decision-making close to the data source, providing fault tolerance and lower latency. Alharbi and Aldossary [2] propose an energy-efficient cloud-to-fog-to-edge architecture for smart agriculture, demonstrating reductions of up to 36% in energy consumption, 43% in carbon emissions, and 86% in network traffic through hierarchical task allocation and the use of low-power communication.

In a real-world deployment scenario, Et-Taibi et al. [5] describe a cloud-to-edge solution applied to soilless drip irrigation systems, in which the integration and distribution of heterogeneous sensors are into distributed processing flows. The results point to gains in optimizing water and energy use.

Focused on long-range, low-energy scenarios, Zhang et al. [12] present a LoRa-based and edge computing system that incorporates a variant of the Penman–Monteith algorithm for evapotranspiration calculation, supports multiple protocols via a gateway, and uses MQTT/WSS communication. Field tests showed a range of up to 4 km with low packet loss, ensuring reliable data transmission. Complementarily, Kasera and Acharjee [7] propose a smart irrigation framework supported by LoRa and edge IoT, integrating a robust transmission scheme over MQTT, efficient topologies for M2M communication, and reinforcement learning elements for dynamic irrigation scheduling.

Edge approaches also stand out in isolation by reducing latency, filtering data close to the source, and allowing for greater local decision-making autonomy. Munir et al. [9] propose a system that combines edge computing, ontologies, and machine learning to classify water needs based on multiple environmental variables, avoiding the transmission of unnecessary data to the backend. Similarly, Ndunagu et al. [10] describe a wireless sensor network integrated with a cloud platform but with pre-processing on the edge server in time windows, employing ML algorithms (e.g., KNN) and achieving high accuracy in irrigation decisions. Although they still present limitations in defining robust synchronization mechanisms with the cloud and reliable disconnected operation policies, these works reinforce the gains of edge computing in terms of autonomy and efficiency.

Taken together, these studies demonstrate the concrete benefits of hybrid architectures, energy efficiency, reduced traffic, expanded coverage, and significant expansion of autonomy in decision-making. However, few studies detail practical protocols for bidirectional replication and mechanisms for eventual consistency between cloud and edge in situations of prolonged disconnection, a recurring challenge in rural areas.

Practical adoption in small farms requires competitive costs, interoperability, and ease of maintenance. [11] develops a low-cost, open-source platform that integrates FIWARE (LD), water balance models, and a *datalogger* compatible with multiple protocols (LPWAN, LoRaWAN, NB-IoT, GSM, etc.), favoring reuse and integration. Complementarily, [8] proposes an ESP32, MQTT, and cloud services system, reporting significant reductions in water consumption and describing sensor calibration procedures along with a structured development process. Although these works advance interoperability and total cost of ownership, most lack formal failover policies, post-failure reconciliation, and consistent replication between layers, which are critical points for environments with unstable power and connectivity.

The review of related works indicates that IoT-based smart irrigation systems have expanded in terms of technological integration, resource efficiency, and decision support. Cloud-centric architectures have demonstrated maturity in terms of scalability, governance, and analytical tools. However, they present important limitations when applied to rural areas subject to connectivity instability. In turn, hybrid cloud-to-edge and cloud-to-fog-to-edge models reinforce the potential to reduce latency, filter data close to the source, and expand local autonomy. They achieve concrete benefits such as greater energy efficiency, lower network traffic, and increased reliability in long-distance scenarios. However, bidirectional replication and eventual consistency mechanisms in situations of prolonged disconnection remain underexplored.

Open and low-cost platforms emerge as promising alternatives for adoption in small farms, combining interoperability, component reuse, and a more accessible total cost of ownership. However, such solutions still lack formal failure policies, post-failure reconciliation, and consistent replication across layers—critical aspects in rural contexts marked by recurring power and connectivity failures.

In general, the reviewed literature demonstrates significant progress but also reveals gaps that motivate this research. Among these, the following stand out: (i) the absence of robust protocols for reliable synchronization between cloud and edge; (ii) the need for effective offline operation policies; and (iii) the lack of practical strategies to ensure data consistency and service continuity in environments subject to infrastructure constraints. These points guide the contribution of this paper, which proposes a resilient architecture for smart irrigation that addresses these challenges.

3 Methodology

The architecture's construction proposes an incremental trajectory for the intelligent security system based on the Design Thinking approach. This approach combines cycles of empathy, problem definition, ideation, prototyping, and validation. Figure 1 shows these cycles, including the four prototyping phases. The project's evolution consists of five major structural phases:

– Immersion and Problem Definition Phase: The project's starting point was a field study conducted on a farm. The main challenges identified are related

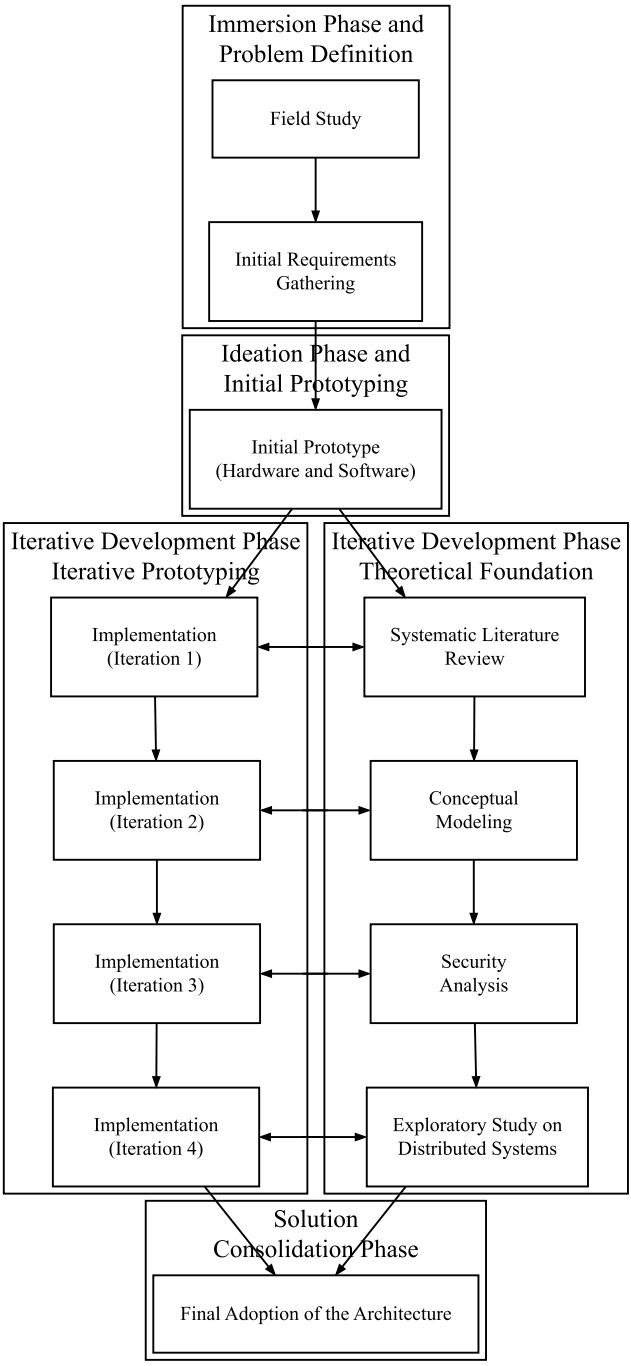

Fig. 1. Work Evolution Flowchart.

to functionality in remote environments and limited connectivity. Since there was direct contact with users and their routines, it was possible to gather initial requirements while considering their real needs and the constraints of the rural context.
- Initial Ideation and Prototyping Phase: With the requirements defined, the project moved on to formulate hypotheses about the system architecture. We explore technical and conceptual alternatives based on empirical observation and a literature review. The initial prototyping aims to test these installations and obtain feedback quickly.
- Iterative Development Phase: Subsequent development occurred iteratively and on two parallel fronts: one focused on building and improving prototypes, and the other focused on the theoretical studies that underpinned architectural decisions.
- Iterative Prototyping: The initial prototype evolved over four consecutive cycles, incorporating improvements based on practical testing and continuous feedback. Each new version serves to validate solutions, identify limitations, and refine the system gradually. In parallel with prototyping, we conduct theoretical studies that provide a basis for the design of the final architecture.
- Solution Consolidation Phase: As the culmination of the process, the final system architecture was defined and adopted. This architecture integrates lessons learned in the field, data from practical tests, and technical fundamentals extracted from the literature. The result is a distributed solution oriented towards edge and cloud layers, prepared to operate resiliently in environments with limited connectivity, without sacrificing scalability, security, and ease of maintenance.

4 Implementation

This section presents a simplified view of how the system works based on user interactions with the resources provided. The web application was developed in a modular structure, using the Django framework to create internal applications (apps) that organize business logic and facilitate maintenance and system expansion. Each application focuses on specific functionalities, allowing a clear separation of responsibilities within the project. The main applications implemented are:

- **Core**: responsible for managing the central entities of the system, bringing together both user and client models, as well as the technical elements of the infrastructure;
- **Telemetry**: manages the reception and storage of data from sensors installed in devices (IoT);
- **Irrigation**: controls the scheduling and activation logic of the actuators responsible for irrigation;
- **Monitoring**: stores connectivity verification data between the cloud and the edge, generating a connection health metric based on the latest data collection.

Although the same Django applications are present in both the edge and cloud instances, their functionalities are adapted to meet the distinct operational contexts of each layer. The C4 level 3 diagrams presented in Fig. 2 which detail the cloud components, and Fig. 3, which represent the edge components, evidence of those functional differences.

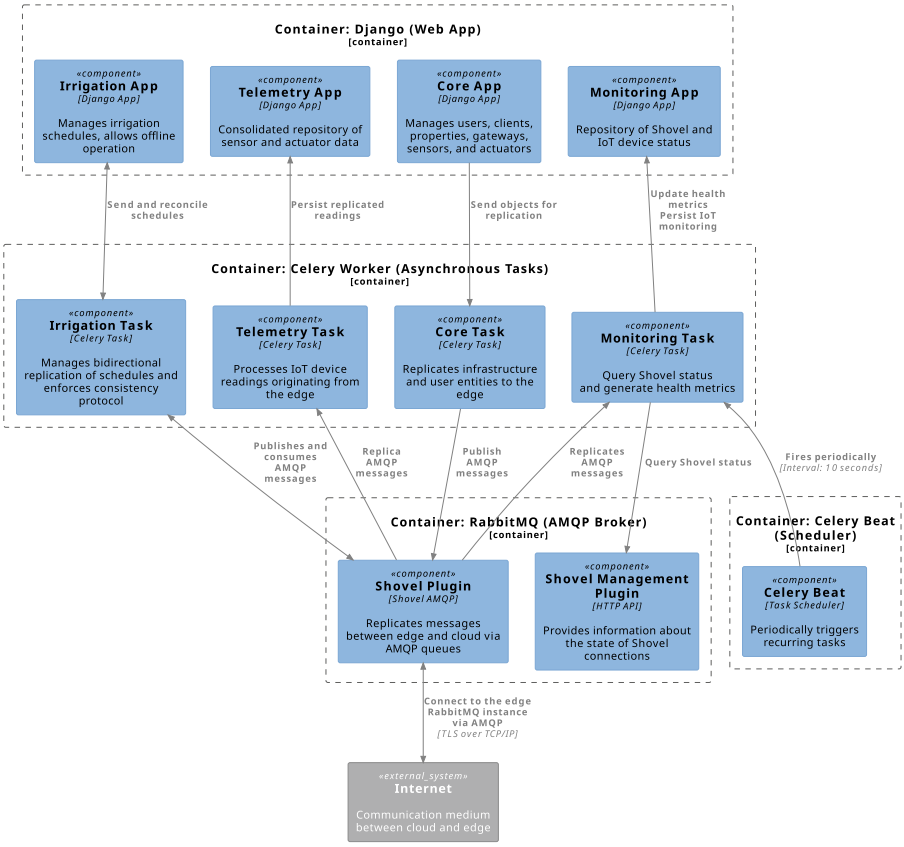

Fig. 2. C4 Level 3—Internal components of the cloud server.

In many cases, this involves data replication via RabbitMQ AMQP message queues, which, in Django, is handled by asynchronous tasks managed with Celery. We detail the replication process in the following sections, which present the specific flows of each application in the system, accompanied by UML sequence diagrams that illustrate the communication between the components.

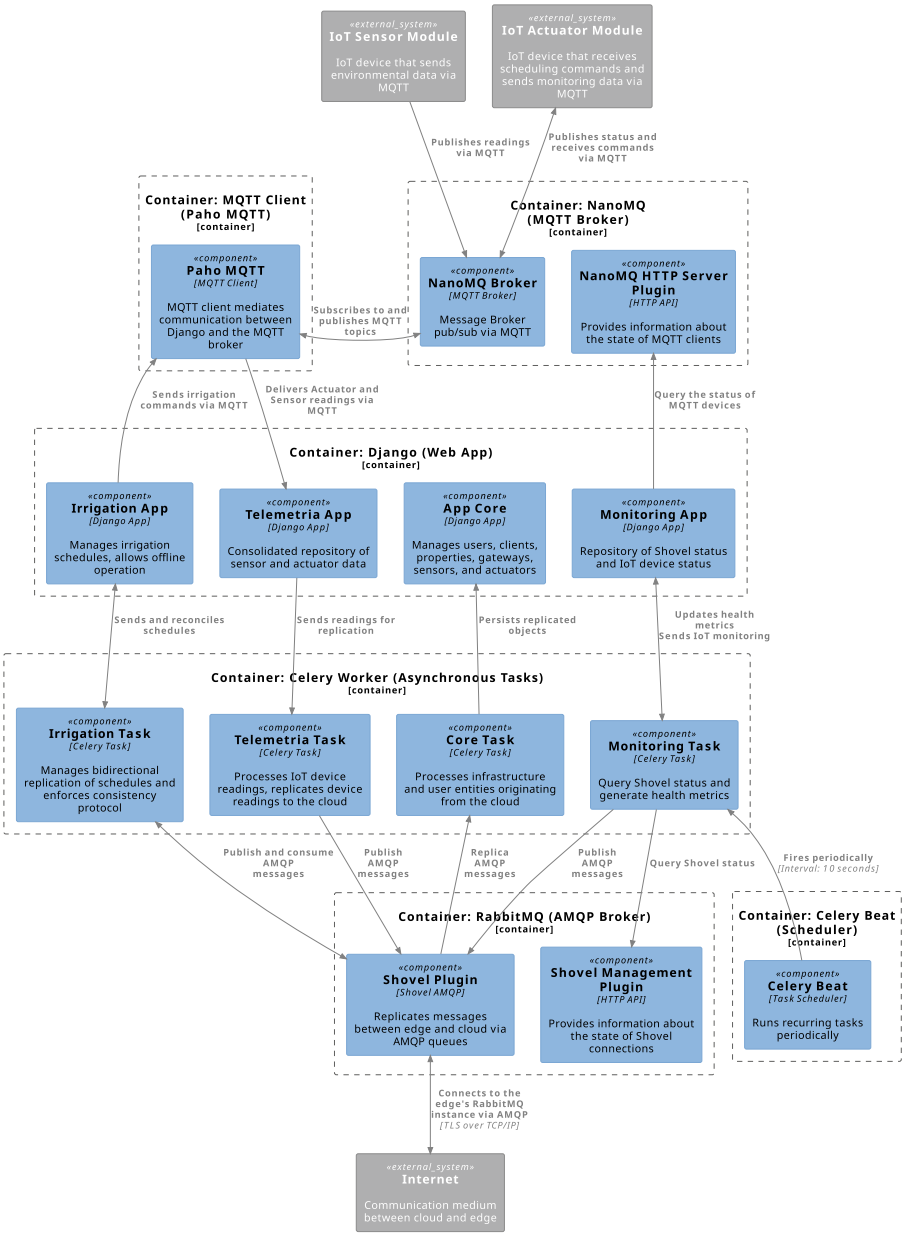

Fig. 3. C4 Level 3—Internal components of the edge server.

4.1 Application Core

The application *core* brings together the central entities of the system, which are responsible for structuring both the organizational model and the technical elements of the infrastructure. Models such as `User` and `Client` represent the different types of users and their links to the system instance, allowing for role-based access control (such as owner, manager, and employee) and logical isolation by client. The models `Property`, `Gateway`, `Actuator`, and `Sensor` represent the technical elements that make up the distributed architecture of the system, connecting IoT devices to the software through a hierarchical structure.

The interdependence between them justifies the decision to cluster these models into a single application: clients associate users who, in turn, manage properties equipped with gateways, sensors, and actuators. Thus, the *core* application serves as a common structural base for the other system modules, consolidating the essential information for its operation.

The replication logic of the *core* follows the cloud-to-edge direction. Since changes to these models are only allowed in the cloud instance, the end user does not directly manipulate this part of the application. This strategy simplifies architecture management by eliminating bidirectional replication and the need for conflict resolution while ensuring the consistency of essential data across both layers of the system. The sequence diagram represents the data flow (Fig. 4).

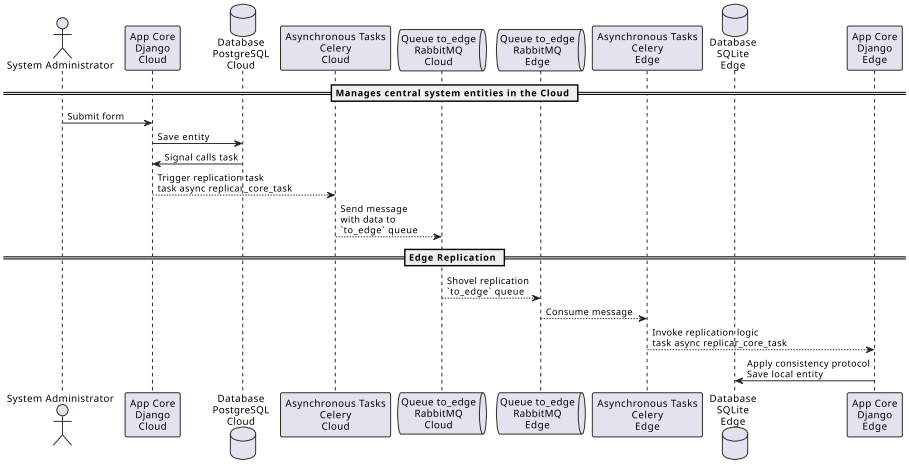

Fig. 4. Sequence—Replication of central system entities (core).

4.2 Telemetry Application

The telemetry application is responsible for managing the flow of data collected by sensors and actuators installed in the field. This flow is unidirectional: IoT

devices send readings via the MQTT protocol to the edge server, where the data is received, processed, and stored locally.

After this initial persistence, the information is asynchronously replicated to the cloud instance using the AMQP protocol, ensuring its centralization, security, and availability for later analysis. End users or system administrators do not directly manipulate those records, since sensors generate data automatically. In this way, the telemetry application acts as an entry point, at the edge, for raw data from the physical environment, and, in the cloud, it serves as a consolidated repository. The sequence diagram represents the data flow (Fig. 5).

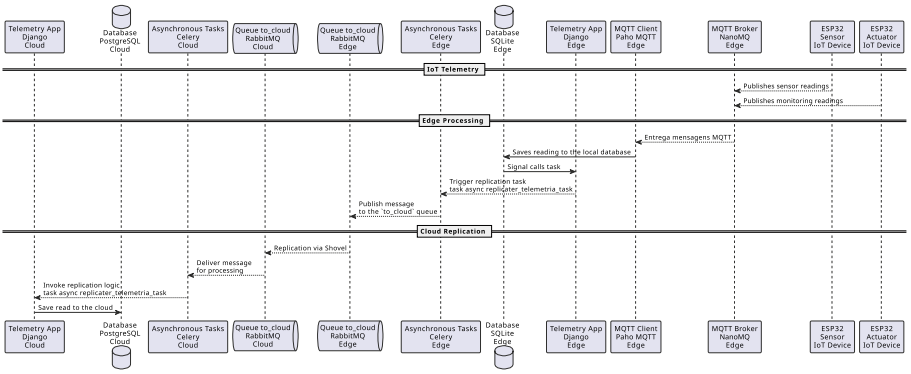

Fig. 5. Sequence—Telemetry reading replication flow.

4.3 Irrigation Application

The irrigation application is responsible for controlling the scheduling and activation logic of the actuators involved in the irrigation process. Since these schedules are defined directly by the end user, the system needs to support disconnected operations, ensuring functionality even in situations of instability or loss of internet connection, as designed in this project. For this reason, the administration of this module for the end user is available in both the cloud instance and the local instance of the edge server.

Under normal conditions, schedule management is through the interface available in the cloud, accessed via the internet, referred to here as "connected operation". The sequence diagram represents the data flow in a connected operation (Fig. 6).

However, in disconnection situations, the system allows local access to the replicated instance at the edge, ensuring operational continuity. That characterizes the situation as a "disconnected operation" [4]. The sequence diagram represents the data flow in a disconnected operation (Fig. 7).

To enable this functionality, the *irrigation* application implements a bidirectional replication flow between the edge and the cloud. Since modifications on

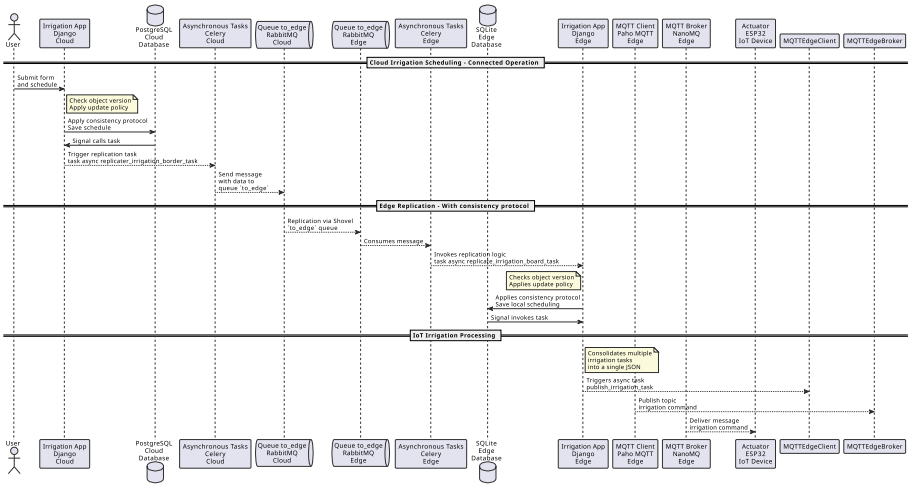

Fig. 6. Sequence—Irrigation scheduling with connected operation.

the same data can be in both instances during a period of disconnection, the system adopts a simplified consistency protocol to handle update conflicts.

The main conflict situation occurs when the same schedule is changed simultaneously at both the edge and in the cloud during disconnection. After reconnection, both sides try to replicate their versions of the same object, which generates divergence. To resolve this impasse in a conflict situation, the protocol defines the prevalence of the edge version, based on the premise that local operators have greater knowledge of the operational situation.

Despite being a simple approach, this policy ensures the functional consistency of the system and reflects the priority of local action in unstable scenarios. In addition, the *irrigation* app, in edge cases, is responsible for consolidating the various registered schedules into a single daily routine command. This command is transmitted to the actuator device via the MQTT protocol, enabling the automated execution of the irrigation plan.

4.4 Monitoring Application

The monitoring application is responsible for informing the end user about the connection status between the edge instance and the cloud. The Shovel plugin, supported by RabbitMQ, performs data replication between these layers. This plugin allows the creation of integrated AMQP queues between different brokers and supports monitoring the connection between instances.

RabbitMQ itself provides, through a management plugin, information about the Shovel connection status. A fixed frequency configured in RabbitMQ itself defines the granularity of this update. Due to the periodic nature of this data, the performance of its collection is managed through a scheduled task with Celery Beat, Celery's recurring task scheduler.

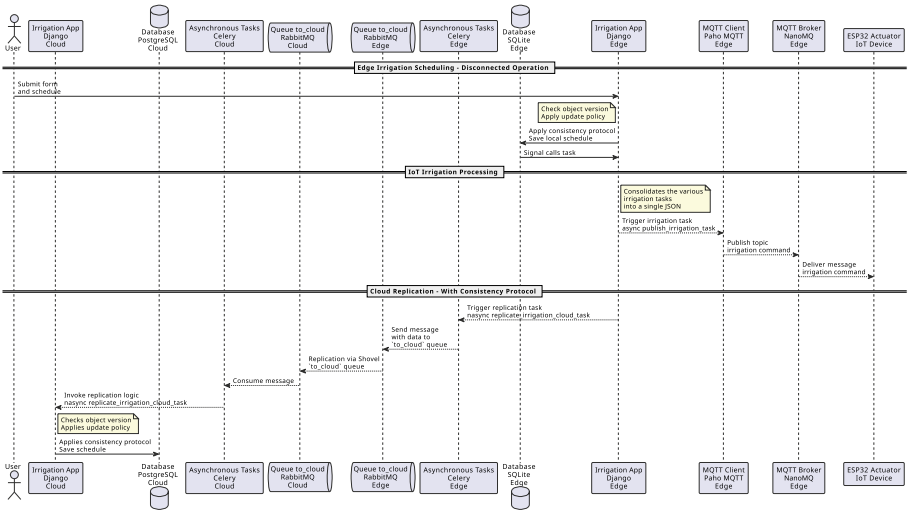

Fig. 7. Sequence—Irrigation schedule with disconnected operation.

The collection interval recommended by the official RabbitMQ documentation, fixed at 10 s, was chosen. The collected information is stored in the *monitoring* application. It is used to generate a connection health metric. This metric is based on the last ten data collections and represents the percentage of successful communication between the cloud and the edge. The metric indicates the stability of communication between the servers and is updated periodically at the same interval as the *Shovel* data collection.

Based on this index, the system classifies the connection status as follows:

- `Disconnected`: when the index is equal to 0;
- `Bad`: when the index is between 1% and 39%;
- `Moderate`: between 40% and 69%;
- `Good`: from 70%.

The following sequence diagram represents the process of monitoring the connection between the edge and the cloud. This flow is executed in both instances—cloud and edge—since the logic for collecting and evaluating the connection is identical in both operational contexts (Fig. 8).

In addition to monitoring edge-to-cloud connections, the *monitoring* application is also responsible for tracking MQTT clients connected to the *NanoMQ* broker via the *http_server* plugin. This functionality is similar to *Shovel* monitoring, except that it runs exclusively on the edge instance, as that is where the installation of *NanoMQ* is and the localization of IoT devices occurs. The information obtained locally is subsequently replicated to the cloud, ensuring unified visibility. As with *Shovel*, the system calculates a health metric based on the presence and stability of MQTT client connections, allowing for an assessment of the overall network health of devices.

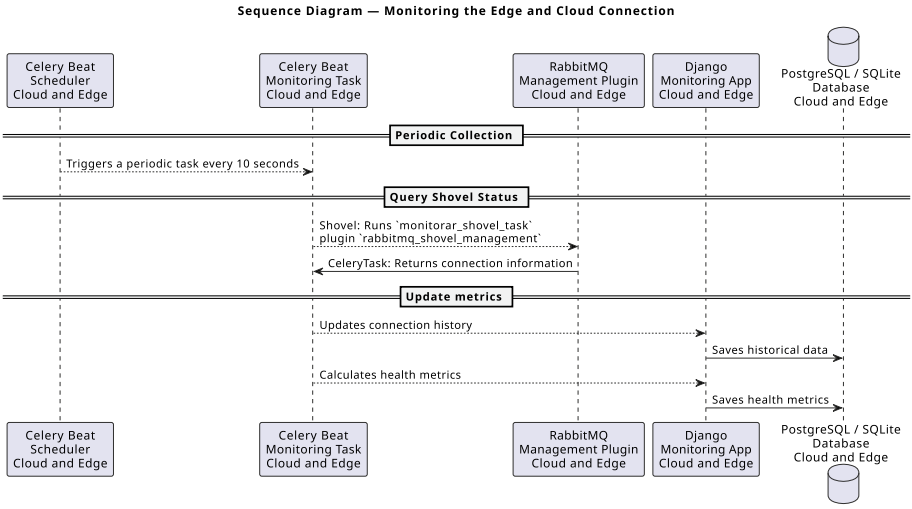

Fig. 8. Sequence—Monitoring the connection between edge and cloud.

Based on this information, the system can automatically switch application access to the local edge instance, ensuring functionality for clients connected to the local network during periods of instability or internet disconnection through the disconnected operation mechanism. Since this is a local diagnostic component, the *monitoring* app does not participate in data replication between the edge and the cloud. Just like the *telemetry* app, the manipulation of logs from this module is not directly by end users or system administrators.

The following sequence diagram represents the process of monitoring MQTT connections from IoT devices to the *NanoMQ* broker. This flow is executed exclusively on the edge instance, as this is where both *NanoMQ* and the connected devices are located. Connectivity information is collected periodically via the *http_server* plugin API, and the data obtained is used to calculate a health metric for the MQTT clients. The information is subsequently replicated to the cloud, allowing for a consolidated view of the network state of the devices (Fig. 9).

5 Experimental Results

First Iteration: The first iteration validated the technical fundamentals of the architecture. It considers field constraints such as limited connectivity, electrical instability, and users' low technological familiarity. The system is divided into three modules: the sensor (ESP32-S3), the edge (Orange Pi with Django/SQLite), and the cloud (Django/SQLite). The sensor publishes data via MQTT to Mosquitto on the edge; the edge, in turn, processes and stores the readings, operating in offline mode. The cloud interacts with the edge via MQTT (HiveMQ), maintaining bidirectional communication. The architecture

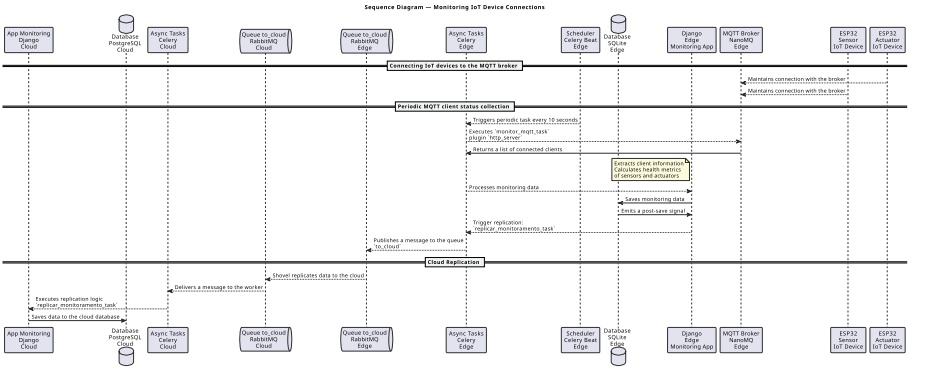

Fig. 9. Sequence—Monitoring MQTT connections at the edge.

is designed to tolerate faults and support subsequent synchronization with the cloud.

Second Iteration: Improvements were introduced in communication and data replication, overcoming the limitations of traditional MQTT. The NanoMQ (edge) and EMQX (cloud) brokers were adopted, with Data Bridge support to ensure message persistence and delivery, even during disconnections. The devices were organized into sensor and actuator modules, using UUID identifiers and JSON structures. NanoMQ was run via Docker on the Orange Pi, eliminating direct dependencies between Django and MQTT.

Third Iteration: This phase focused on the security of the solution. We used Docker, Gunicorn, and Nginx containers on the servers, with secure communications via TLS, authentication, and ACLs. The ESP32-S3 has been equipped with embedded protection mechanisms, including Secure Boot, eFuses, and Flash Encryption. We incorporated robust sensors and solid-state relays to ensure durability, and WPA2/WPA3 protects Wi-Fi communication. We also considered the use of other technologies in future versions. To ensure energy redundancy, we implemented a UPS and batteries recharged by solar panels, in addition to physical protections for the devices.

Fourth Iteration: Communication between the edge and the cloud has been redesigned to use **AMQP queues** with **RabbitMQ**, replacing the previous use of MQTT brokers for replication. This change has enhanced decoupling and increased system resilience. For task orchestration, **Celery** has been integrated with a **Redis** backend, which allows for asynchronous and scalable execution of tasks. Despite the change, **NanoMQ** was maintained at the edge for MQTT communication with IoT devices. An MQTT client (Paho-MQTT) was encapsulated in a Celery worker, ensuring that sending and receiving messages did not affect the availability of the Django application.

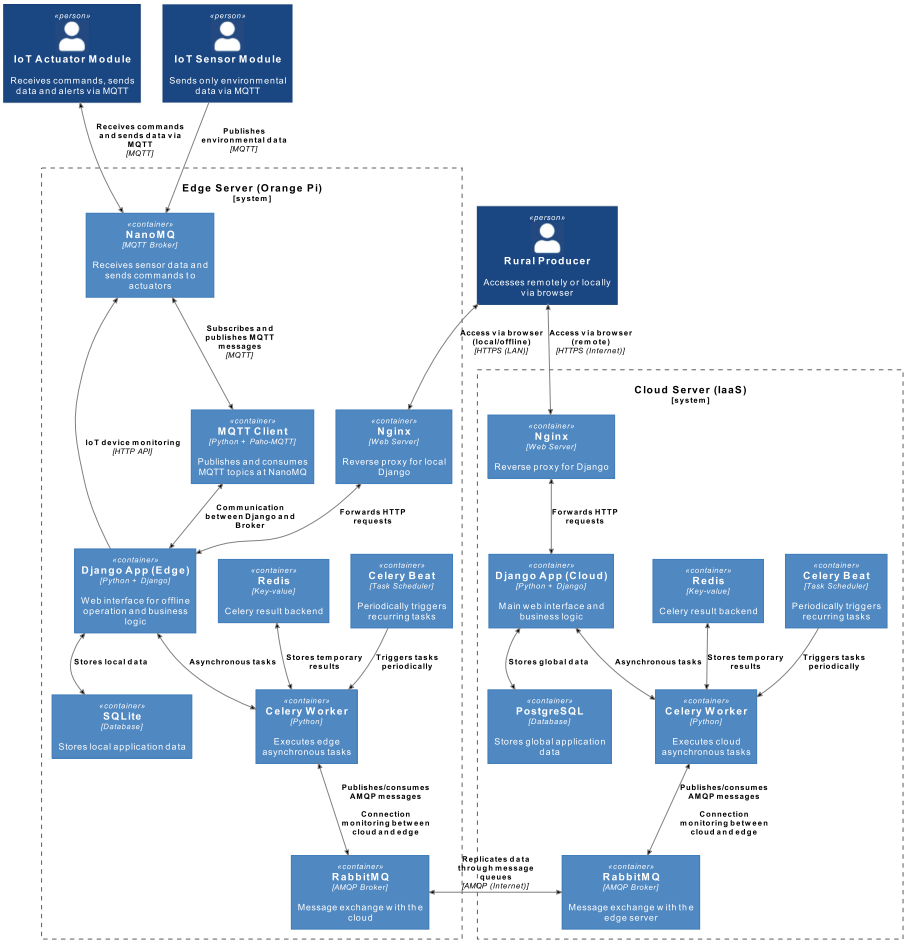

Fig. 10. Solution Consolidation.

Solution Consolidation: The final architecture consolidates the theoretical and practical learnings into a distributed, resilient, secure, and scalable solution, structured in edge and cloud layers. This structure allows for continuous operation even during connectivity or power failures. Figure 10 shows the container diagram of the consolidated architecture.

We conduct Constant Request Rate (CRR) experiments to evaluate the performance of the proposed architecture in real-world operations. We implement the tests using a custom Django command (BaseCommand) capable of generating HTTP requests to create schedules in the API using pseudo-random values. The interval between requests was used as a load parameter, allowing for the measurement of the system's saturation limit and the impact of the storage hardware on the processing rate.

Table 1. Average throughput (msg/s) for different devices and request intervals

Device/Interval	50 ms	100 ms	250 ms
eMMC	5.02	3.87	3.48
SD Card	4.42	3.21	2.98
SSD NVMe	6.87	5.92	5.31

Table 2. Maximum observed throughput (msg/s) in CRR tests

Device/Interval	50 ms	100 ms	250 ms
eMMC	12.00	10.50	6.50
SD Card	10.23	8.97	5.41
SSD NVMe	17.32	14.65	12.44

We selected the scheduling feature because it requires local writing and bidirectional replication between the edge and the cloud. Each execution consisted of 1000 requests under three different emission rates: 250 ms, 100 ms, and 50 ms. All tests were run on the same Orange Pi 3B edge server with 4 GB of RAM, varying only the storage device among NVMe SSD, eMMC, and SD card.

The objective was to measure the influence of the storage type and request rate on system throughput. The devices represent three distinct categories of use in edge computing: low cost (SD), intermediate (eMMC), and high-performance (NVMe). Each scenario automatically stored the total execution time, average throughput, maximum throughput, and complete logs in CSV format. Replication to the cloud occurred after the persistence of records in the local database, allowing for the measurement of the direct effect of writing on the hardware.

We analyze the following metrics:

- Average throughput (msg/s) – sustained processing rate;
- Maximum throughput (msg/s) – highest recorded peak;

Average latency, p50, p95, and p99 were also recorded for complementary analysis.

Table 1 presents the average throughput per device and interval. The NVMe SSD delivered the best performance in all scenarios, with an average of 36% better than eMMC and 55% superior to the SD card.

In the most intense scenario (50 ms), NVMe reached 6.87 msg/s, while eMMC and SD were below 5 msg/s. At longer intervals, the limit became the time between requests, reducing the impact of the hardware. Table 2 shows the observed peaks. The results indicate that NVMe maintains greater stability under load, showing a smaller distance between the average and peak, while SD and eMMC accumulate backlog at shorter intervals. Storage hardware directly influences the event processing rate at the edge and the replication speed to the cloud.

6 Conclusion

The development of this smart irrigation system highlighted the complexity involved in building resilient solutions for family farming in environments with limited infrastructure. More than a simple connection between devices and servers, the project required addressing challenges such as data replication and consistency, tolerance for connectivity failures, and orchestration of asynchronous tasks between edge and cloud.

The iterative approach provides practical validation of each component, promoting the continuous evolution of the architecture. It consolidates a distributed, secure, and scalable model. The resulting system is capable of operating even in the face of power and connection failures. The developed solution represents significant progress for the Agriculture 4.0 scenario by democratizing access to technologies restricted to large farms. It proposes a viable and accessible alternative for small producers. Furthermore, it establishes a solid foundation for future expansions.

Future work includes evaluating the system's usability with end users. It covers automating the irrigation cycle based on environmental conditions and crop characteristics.

References

1. Abbasi, R., Martinez, P., Ahmad, R.: The digitization of agricultural industry – a systematic literature review on agriculture 4.0. Smart Agric. Technol. **2**, 100042 (2022). https://doi.org/10.1016/j.atech.2022.100042. https://www.sciencedirect.com/science/article/pii/S2772375522000090
2. Alharbi, H.A., Aldossary, M.: Energy-efficient edge-fog-cloud architecture for IoT-based smart agriculture environment. IEEE Access **9**, 110480–110492 (2021). https://doi.org/10.1109/ACCESS.2021.3101397
3. Boursianis, A.D., et al.: Smart irrigation system for precision agriculture—the ARETHOU5A IoT platform. IEEE Sens. J. **21**(16), 17539–17547 (2021). https://doi.org/10.1109/JSEN.2020.3033526
4. Coulouris, G., Dollimore, J., Kindberg, T., Blair, G.: Distributed Systems: Concepts and Design, 5th edn. Addison-Wesley, São Paulo (2013)
5. Et-taibi, B., Abid, M., mahjoub Boufounas, E., Benhaddou, D.: Cloud and edge based smart agriculture: a real-world deployment. In: 2024 International Conference on CircuiCloud and Edge based Smart Agriculture: A Real-World Deployment, Systems and Communication (ICCSC), pp. 1–6 (2024). https://doi.org/10.1109/ICCSC62074.2024.10617136
6. Kamienski, C., et al.: Smart water management platform: IoT-based precision irrigation for agriculture †. Sensors **19** (2019). https://doi.org/10.3390/s19020276
7. Kasera, R.K., Acharjee, T.: A comprehensive IoT edge based smart irrigation system for tomato cultivation. Internet Things **28**, 101356 (2024). https://doi.org/10.1016/j.iot.2024.101356
8. Morchid, A., Alblushi, I.G.M., Khalid, H.M., Alami, R.E., Sitaramanan, S.R., Muyeen, S.M.: High-technology agriculture system to enhance food security: a concept of smart irrigation system using internet of things and cloud computing. J. Saudi Soc. Agric. Sci. (2024). https://doi.org/10.1016/j.jssas.2024.02.001

9. Munir, M., Bajwa, I.S., Ashraf, A., Anwar, W., Rashid, R.: Intelligent and smart irrigation system using edge computing and IoT. Complex. **2021**, 6691571:1–6691571:16 (2021). https://doi.org/10.1155/2021/6691571
10. Ndunagu, J., Ukhurebor, K., Akaaza, M., Onyancha, R.B.: Development of a wireless sensor network and IoT-based smart irrigation system. Appl. Environ. Soil Sci. (2022). https://doi.org/10.1155/2022/7678570
11. Puig, F., Díaz, J.A.R., Soriano, M.: Development of a low-cost open-source platform for smart irrigation systems. Agronomy (2022). https://doi.org/10.3390/agronomy12122909
12. Zhang, Y., et al.: Research and development of an IoT smart irrigation system for farmland based on lora and edge computing. Agronomy (2025). https://doi.org/10.3390/agronomy15020366