# On the Scale Transition of Event-Driven IoT Architectures: An Experimental Evaluation

**Cairo Mateus Neves Ribeiro**[1] **, Júlio Cezar Estrella**[1]

[1] Institute of Mathematics and Computer Sciences
University of São Paulo (USP)
São Carlos – SP – Brazil

cairo.ribeiro@usp.br, jcezar@icmc.usp.br

***Abstract.*** *Event-driven Internet of Things (IoT) architectures are widely adopted due to their flexibility and decoupling properties, yet their behavior during scale transitions remains insufficiently understood. In particular, how performance and reliability evolve when systems move from moderate to high workload regimes is often underexplored. This paper presents an experimental evaluation of an event-driven IoT architecture focusing on the transition from medium- to large-scale operation. The proposed architecture combines MQTT and Apache Kafka to decouple data ingestion from processing and is evaluated using a reproducible testbed with synthetic workloads under constant and exponential load patterns. Experimental results show that HTTP latency remains stable across all scenarios, while message acceptance rates reveal clear throughput saturation points under extreme load. These findings demonstrate that scale transition manifests primarily through reliability degradation rather than latency increase, highlighting the importance of multi-dimensional evaluation when assessing scalability in IoT systems.*

***Resumo.*** *Arquiteturas de Internet das Coisas (IoT) baseadas em eventos são amplamente utilizadas devido à sua flexibilidade e ao desacoplamento entre ingestão e processamento de dados. No entanto, o comportamento desses sistemas durante transições de escala ainda é pouco compreendido, especialmente no que se refere à evolução de desempenho e confiabilidade sob cargas elevadas. Este trabalho apresenta uma avaliação experimental de uma arquitetura IoT orientada a eventos, com foco na transição de escala de ambientes de médio para grande porte. A arquitetura proposta integra MQTT e Apache Kafka e é avaliada por meio de um testbed reprodutível, utilizando cargas sintéticas constantes e exponenciais. Os resultados experimentais indicam que a latência HTTP permanece estável em todos os cenários, enquanto a taxa de aceitação de mensagens evidencia pontos claros de saturação sob cargas extremas. Esses resultados mostram que a transição de escala se manifesta principalmente pela degradação da confiabilidade, e não pelo aumento da latência, reforçando a necessidade de avaliações multidimensionais de escalabilidade em sistemas IoT.*

## 1. Introduction

Internet of Things (IoT) systems are commonly developed through small-scale prototypes, often deployed in controlled laboratory environments. While such prototypes are

essential for validating functionality and feasibility, they rarely expose the challenges that emerge when systems evolve toward larger-scale deployments. As the number of devices, message rates, and processing stages increase, architectural decisions that are adequate at small scale may lead to performance degradation, instability, or loss of reliability. Understanding how IoT architectures behave across scale transitions therefore remains a relevant problem in distributed and networked systems.

A common approach to address scalability in IoT relies on event-driven designs, typically combining lightweight protocols at the edge with distributed messaging platforms in the core. Protocols such as MQTT enable efficient communication with constrained devices, while streaming backbones such as Apache Kafka provide high-throughput and decoupled data dissemination. Although these technologies are widely adopted, their combined behavior across different deployment scales is often assumed rather than systematically evaluated. In practice, many studies focus either on architectural descriptions or on single-scale benchmarks, leaving a gap in experimental analyses that explicitly investigate how performance evolves as the system transitions from small to medium and larger scales.

This paper addresses this gap by experimentally studying the scale transition of an event-driven IoT architecture. Instead of validating a specific application scenario, we focus on system-level behavior under increasing load conditions. Our goal is to analyze how latency, throughput, and robustness metrics evolve as the architecture moves beyond its initial operating regime. To this end, we implement a medium-scale prototype that integrates MQTT-based edge communication with a Kafka-based streaming backbone and containerized backend services. The prototype is evaluated using controlled workloads that emulate increasing numbers of virtual devices and message rates.

The experimental evaluation follows two complementary workload patterns. First, constant-load experiments are used to characterize steady-state behavior and identify saturation trends. Second, exponentially increasing workloads stress the system under burst-like conditions, revealing backpressure effects and tail latency behavior. Results are reported using distribution-aware metrics, including boxplots and percentile-based statistics, to capture variability and extreme cases that are critical in distributed systems.

The contributions of this paper are threefold: (i) an explicit formulation of the scale transition problem in event-driven IoT architectures; (ii) a reproducible experimental evaluation that analyzes system behavior under constant and increasing workloads using distribution-based metrics; and (iii) empirical insights into performance degradation patterns and robustness limits as load intensifies. Together, these contributions provide practical evidence to support architectural decisions when evolving IoT systems beyond small-scale prototypes.

The remainder of this paper is organized as follows. Section 2 reviews related work on scalable IoT and event-driven architectures. Section 3 presents the adopted system architecture and its scale-oriented design. Section 4 describes the experimental methodology. Section 5 reports the experimental results. Section 6 discusses implications and limitations, Section 7 discusses the main threats to validity associated with the experimental evaluation and Section 8 concludes the paper and outlines future work.

## 2. Related Work

Scalability remains a central concern in Internet of Things (IoT) systems, particularly as deployments evolve from small prototypes to operational environments with higher device counts, increased event rates, and more complex data pipelines. Foundational surveys consolidate the main architectural challenges of IoT platforms, including heterogeneity, interoperability, and data management. Daissaoui et al. [Daissaoui et al. 2020] survey IoT and analytics for smart buildings, while Ngu et al. [Ngu et al. 2017] review middleware issues and enabling technologies. Although these works are valuable to frame the problem space, they are primarily conceptual and do not explicitly characterize how performance and reliability evolve across *scale transitions*.

A common approach to address scalability is the adoption of cloud–edge and fog computing paradigms, which distribute computation closer to data sources to reduce latency and network overhead. Comprehensive surveys discuss architectural options, resource management challenges, and the role of edge and fog layers in large deployments [Khan et al. 2020, Khan et al. 2019, Yousefpour et al. 2019, Bonomi et al. 2022, Ghobaei-Arani et al. 2020]. While these studies provide strong architectural motivation for distributed and event-driven designs, experimental evaluations are typically limited to a fixed deployment scale or static workloads, which may obscure saturation effects and tail behavior that emerge under aggressive load growth.

Event-driven architectures are widely employed to improve decoupling and throughput in IoT systems. At the edge, MQTT is commonly adopted due to its low overhead and suitability for constrained devices [OASIS 2019]. At the core, streaming and dataflow paradigms enable asynchronous processing and buffering, allowing producers and consumers to scale independently. Foundational works discuss data streams and complex event processing [Cugola and Margara 2012, Margara and Rabl 2019], while surveys analyze elasticity and architectural concerns in distributed stream processing [de Assunção et al. 2017]. In the IoT context, Happ et al. [Happ et al. 2017] evaluate open publish/subscribe platforms against IoT requirements, and Cruz et al. [da Cruz et al. 2018] propose a reference model for IoT middleware. These studies emphasize modularity and interoperability, but they do not explicitly quantify how reliability degrades as systems cross workload regimes.

Experimental studies that evaluate end-to-end IoT pipelines under increasing load remain comparatively scarce. Khriji et al. [Khriji et al. 2022] design and implement a cloud-based event-driven architecture for real-time processing in wireless sensor networks, reporting latency and throughput behavior under realistic conditions. Varatharaj [Varatharaj 2024] discusses scalable event-driven architectures for real-time data processing, focusing on distributed processing characteristics and throughput-oriented evaluation. In parallel, recent works highlight the impact of software architecture and container orchestration on IoT system performance [Freire et al. 2024, Curasma et al. 2024]. However, these efforts typically evaluate a single workload profile or do not explicitly contrast steady-state (constant) and bursty (exponential) conditions, which is critical to reveal throughput saturation and message acceptance limits during scale transitions.

In contrast, this paper explicitly investigates the *scale transition* of an event-driven IoT architecture by combining (i) constant-load experiments for steady-state characterization and (ii) exponential-load experiments to expose burst-driven saturation. By jointly

**Table 1. Comparison of representative related works with respect to scale, workload, and experimental evaluation.**

| Work | Scale | Workload | Metrics | Focus |
|---|---|---|---|---|
| Daissaoui et al. (2020) | Survey | – | – | IoT analytics for smart buildings |
| Ngu et al. (2017) | Survey | – | – | IoT middleware overview |
| Khan et al. (2020) | Survey | – | – | Edge computing for smart cities |
| Yousefpour et al. (2019) | Survey | – | – | Fog and edge paradigms |
| Bonomi et al. (2022) | Conceptual | – | – | Fog computing in IoT |
| Happ et al. (2017) | Small | Static | Throughput | Pub/sub platforms for IoT |
| Cruz et al. (2018) | Conceptual | Static | Qualitative | IoT middleware reference model |
| Khriji et al. (2022) | Medium | Event-driven | Latency, throughput | Cloud-based WSN processing |
| Varatharaj (2024) | Medium | Event-driven | Throughput | Scalable event-driven architecture |
| Freire et al. (2024) | Medium | Controlled | Performance | Container orchestration impact |
| Curasma et al. (2024) | Medium | Controlled | Scalability | Agents and ontologies for IoT |
| **This work** | Medium | Const. + exp. | Latency dist., acceptance | Scale transition analysis |

reporting distribution-aware latency metrics and end-to-end message acceptance rates, we provide empirical evidence that complements survey-based and architecture-centric works. Table 1 summarizes representative literature and positions this study with respect to evaluation scale, workload characteristics, metrics, and experimental focus.

## 3. System Architecture

This section describes the event-driven IoT architecture adopted in this study, which serves as the experimental platform for evaluating scale transition behavior. The architecture is not proposed as a novel design, but rather as a representative and realistic integration of widely adopted technologies, enabling controlled experimentation under increasing workloads. Figure 1 presents a deployment-oriented schematic view of the experimental platform, highlighting the main data path from edge devices to backend services. The figure should be interpreted as an operational view of the system rather than a formal architectural notation, aiming to clarify component interactions and data flow.

At a high level, the architecture is organized into three logical layers: edge devices, messaging backbone, and backend services. This layered organization supports loose coupling between components and allows the system to be progressively stressed without modifying its core structure.

### Edge Layer

The edge layer consists of IoT devices responsible for sensing and publishing data events. In the experimental setup, physical devices are complemented by virtual publishers that emulate large numbers of sensors. All devices communicate using the MQTT protocol, chosen due to its lightweight nature and widespread adoption in IoT deployments. Topics are hierarchically organized to represent different sensing modalities and device identifiers, enabling scalable publish–subscribe interactions.

### Messaging Backbone

To decouple edge communication from backend processing, the architecture employs an event-driven messaging backbone. MQTT messages published by devices are forwarded to a Kafka-based streaming platform through a protocol bridge. This bridge acts as a translation and buffering component, mapping MQTT topics to Kafka topics while preserving message ordering semantics at the topic level. Apache Kafka is used as the central

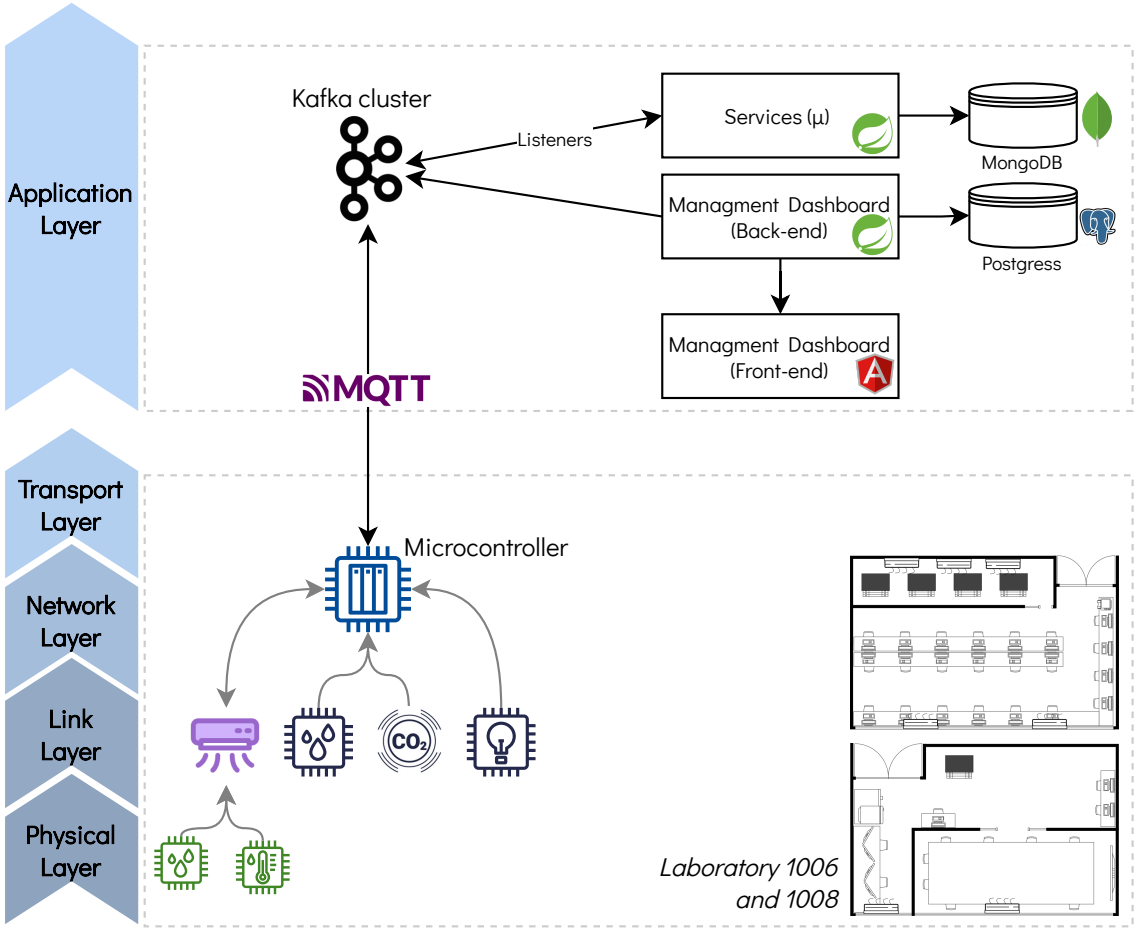


**Figure 1. Overview of the event-driven IoT architecture used as experimental platform, integrating MQTT-based edge communication with a Kafka-based streaming backbone and containerized backend services.**

event bus due to its support for high-throughput ingestion, partitioned topics, and asynchronous consumption. By leveraging Kafka, the architecture enables multiple backend services to consume data streams independently, facilitating scalability and fault isolation. This design choice is critical for evaluating how message rates and processing pipelines behave as the number of producers and events increases.

### Backend Services and Observability

Backend services are implemented as containerized components that subscribe to Kafka topics and perform data persistence and analysis. These services include consumers responsible for storing sensor data and monitoring system health. All backend components are deployed using container orchestration, allowing controlled resource allocation and repeatable experimental conditions. To support experimental evaluation, observability mechanisms are integrated into the backend layer. In particular, Prometheus is used to collect runtime metrics, while Grafana dashboards are employed to visualize HTTP latency, message throughput, and MQTT acceptance rates across the processing pipeline. These metrics enable fine-grained monitoring of the system during workload execution and support the post-hoc analysis presented in Section 5.

This instrumentation enables fine-grained analysis of system behavior under different workload patterns, particularly during saturation and overload conditions.

**Scale-Oriented Design Considerations**

The architecture is explicitly designed to support scale transition experiments. Rather than modifying the architecture itself, scalability is explored by progressively increasing the number of publishers and message rates. This approach isolates the impact of load variation from architectural changes, allowing performance degradation patterns to be directly attributed to scale effects. By combining lightweight edge communication with a high-throughput streaming backbone, the architecture reflects common design practices in contemporary IoT systems. At the same time, its modular structure and observability support make it suitable for systematic experimental analysis, as presented in the following sections.

## 4. Experimental Methodology

This section describes the experimental methodology adopted to evaluate the proposed event-driven IoT architecture under controlled and reproducible conditions. The goal of the experiments is to assess how the architecture behaves when subjected to increasing workloads, focusing on latency stability, throughput, and message acceptance as the system transitions across different load regimes.

The methodology follows a structured approach aligned with experimental best practices in computer networks and distributed systems, comprising: (i) definition of the experimental environment, (ii) workload design, (iii) performance metrics, and (iv) execution procedure. This organization aims to ensure transparency, reproducibility, and fair comparison with related experimental studies.

### 4.1. Experimental Environment

The experimental environment was deployed on a private cloud infrastructure and designed to emulate a medium-scale IoT deployment with multiple concurrent publishers, asynchronous messaging, and centralized stream processing. All components were containerized to reduce environmental variability and facilitate replication. [1]

The environment consists of independent virtual machines hosting load generators, messaging infrastructure, data processing services, and monitoring tools. Communication between components follows the event-driven pipeline introduced in Section 3, combining HTTP-based ingestion, MQTT messaging, and Apache Kafka as the backbone for data flow orchestration.

Table 2 summarizes the main components, technologies, and configurations used in the experimental setup.

### 4.2. Workload Design

Two workload patterns were defined to evaluate system behavior under distinct operational conditions. The first corresponds to a constant-load regime, intended to assess stability and baseline performance. The second introduces an exponential load growth, stressing the system and exposing scalability limits.

[1] Supplementary artifacts related to the experimental platform and workload generation are publicly available at `https://github.com/Smart-LaSDPC/smartlab` and `https://github.com/Smart-LaSDPC/synthesis-data-flows-small-medium-large-iot-architecture`. These repositories provide implementation details and data-flow configurations used in the experiments.

**Table 2. Experimental environment configuration**

| Component | Technology | Configuration / Role |
|---|---|---|
| Load generator | k6 v0.43.0 | HTTP and MQTT workloads; synchronized execution across VMs |
| Virtual machines | Ubuntu 24.04 LTS | 10 VMs, 4 vCPUs, 16 GB RAM each |
| MQTT broker | Eclipse Mosquitto | Message ingestion from IoT publishers |
| Stream platform | Apache Kafka | 4 brokers, asynchronous event processing |
| Coordination service | Zookeeper | Cluster coordination and topic management |
| Application services | Java microservices | Topic control, data persistence, orchestration |
| Databases | PostgreSQL / MongoDB | Metadata storage and sensor data persistence |
| Monitoring | Prometheus + Grafana | Latency, throughput, and system-level metrics |

In both cases, workloads were generated concurrently from multiple virtual machines to emulate distributed IoT publishers. Each VM executed identical scripts, ensuring homogeneous load distribution and synchronized start times.

The evaluated workload scenarios are formally defined and summarized in Table 3.

**Table 3. Workload scenarios evaluated in the experiments**

| Scenario | Load Type | Requests / VM | Growth Pattern | Purpose |
|---|---|---|---|---|
| 1 | Constant | 10 | Fixed | Baseline behavior |
| 2 | Constant | 100 | Fixed | Moderate load |
| 3 | Constant | 1 000 | Fixed | Sustained operation |
| 4 | Constant | 5 000 | Fixed | High concurrency |
| 5 | Constant | 10 000 | Fixed | Saturation analysis |
| 6 | Exponential | 2^x | Doubling every 20 s | Stress testing |

### 4.3. Performance Metrics

The evaluation focuses on end-to-end performance and reliability indicators that are directly relevant to event-driven IoT systems. Metrics were selected to capture both user-perceived responsiveness and system-level robustness under load.

Table 4 presents the metrics considered, their definition, and their relevance to the evaluation.

**Table 4. Performance metrics considered in the evaluation**

| Metric | Description |
|---|---|
| HTTP latency | End-to-end response time for HTTP ingestion requests |
| 95th percentile latency (p95) | Tail latency capturing worst-case user experience |
| MQTT acceptance rate | Ratio of successfully processed MQTT messages |
| Throughput | Number of messages processed per second |
| Confidence interval (95%) | Statistical stability of measured latencies |

In addition, we define MQTT acceptance rate as the ratio of messages that are successfully delivered to the Kafka consumer (i.e., consumed from the corresponding topic) over the total number of MQTT publish attempts generated by the workload. In our setup, the MQTT pipeline is configured in a best-effort manner (no end-to-end retransmission

across the bridge), so acceptance rate captures saturation effects in the MQTT–Bridge–Kafka path rather than application-level retries.

### 4.4. Execution Procedure

Each experimental scenario was executed three times to reduce transient effects and measurement noise. Load generators were synchronized using scheduled execution to ensure consistent start times across all VMs. Metrics were continuously collected by Prometheus and visualized through Grafana dashboards for post-processing and analysis.

This methodology enables a controlled evaluation of how the proposed architecture responds to increasing load, providing a solid basis for the analysis presented in the following section.

## 5. Results

This section presents the experimental results obtained from the evaluation of the proposed event-driven IoT architecture under constant and exponential load conditions. The objective is to analyze latency stability, variability under stress, and message acceptance behavior as the system transitions to higher workload regimes. Results are organized into three parts: (i) performance under constant load, (ii) behavior under exponential load, and (iii) end-to-end reliability measured through MQTT message acceptance rates.

### 5.1. Performance Under Constant Load

We first analyze system behavior under constant request rates, aiming to evaluate baseline stability and scalability without abrupt workload variations. Five scenarios were executed, ranging from low to high concurrency, as defined in Section 4.

Figure 2 presents the distribution of HTTP response times for each constant-load scenario using boxplots. The results show a tight concentration of response times across all scenarios, indicating consistent performance even as the request volume increases.

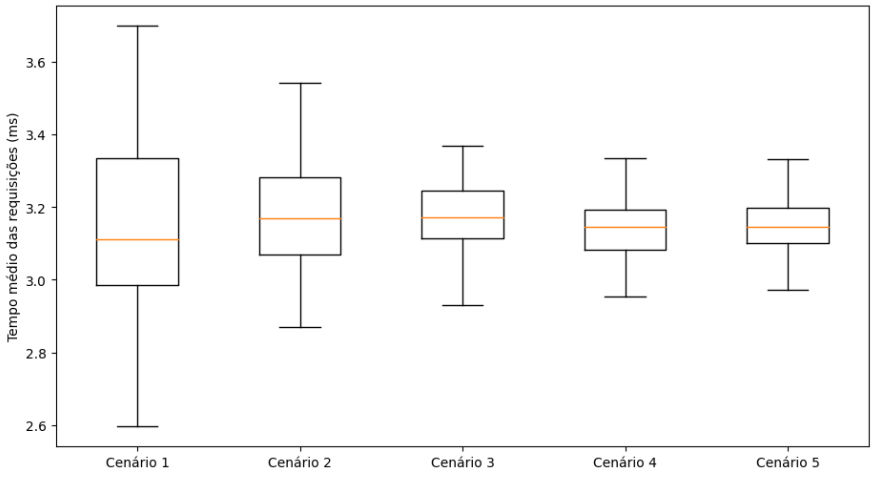


**Figure 2. Response time distribution for constant-load scenarios (HTTP latency boxplots).**

Across all scenarios, the median latency remains close to 3 ms, with limited dispersion. Notably, the interquartile ranges decrease slightly in higher-load scenarios, suggesting improved cache utilization and steady-state behavior of the processing pipeline.

Table 5 reports detailed descriptive statistics for the constant-load experiments, including minimum, maximum, mean latency, standard deviation, and 95% confidence intervals.

**Table 5. Detailed statistics for constant-load scenarios.**

| Scenario | Min (ms) | Max (ms) | Mean (ms) | Std Dev (ms) | 95% CI (ms) |
|---|---|---|---|---|---|
| 1 | 2.06 | 4.85 | 3.27 | 0.68 | [1.95, 4.60] |
| 2 | 1.90 | 7.89 | 3.21 | 0.67 | [2.71, 3.72] |
| 3 | 1.62 | 13.05 | 3.17 | 0.09 | [2.99, 3.36] |
| 4 | 1.52 | 19.94 | 3.14 | 0.08 | [2.97, 3.31] |
| 5 | 1.48 | 43.34 | 3.15 | 0.09 | [2.97, 3.33] |

The results confirm that the architecture sustains stable latency with minimal variance as load increases. Although maximum latency grows with concurrency, the mean and confidence intervals remain nearly constant, indicating resilience to increasing request rates.

### 5.2. Performance Under Exponential Load

The second experiment evaluates system behavior under exponentially increasing workloads, designed to expose scalability limits and stress the messaging and processing pipeline.

Figure 3 shows the response time distributions for the exponential-load scenarios. Compared to the constant-load case, a modest increase in dispersion is observed, particularly in higher scenarios, yet the overall latency remains within a narrow range.

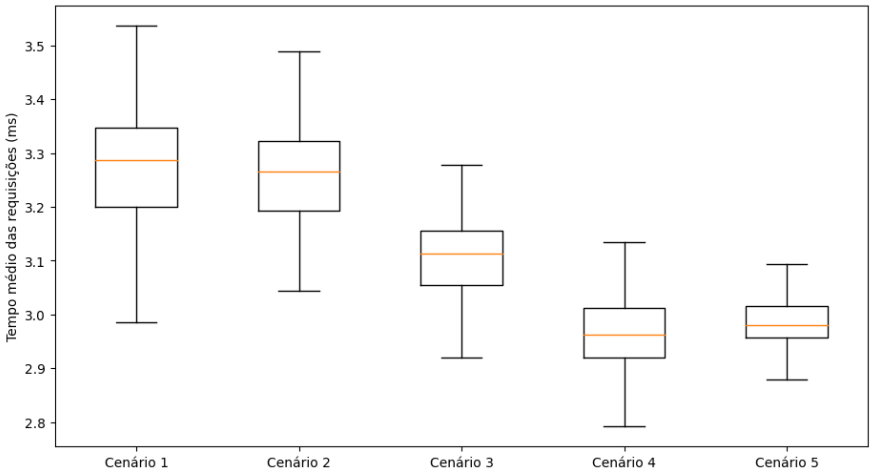


**Figure 3. Response time distribution for exponential-load scenarios (HTTP latency boxplots).**

Despite the aggressive growth pattern, the median and upper quartiles remain stable, suggesting that the event-driven architecture absorbs load bursts without severe degradation in responsiveness.

Table 6 summarizes the detailed statistics for the exponential-load experiments.

Even at the highest stress level, the mean latency remains below 3.1 ms, with limited variation. This result indicates that the architecture maintains predictable performance characteristics under bursty workloads typical of large-scale IoT deployments.

**Table 6. Detailed statistics for exponential-load scenarios.**

| Scenario | Min (ms) | Max (ms) | Mean (ms) | Std Dev (ms) | 95% CI (ms) |
|---|---|---|---|---|---|
| 1 | 2.35 | 4.71 | 3.27 | 0.11 | [3.05, 3.49] |
| 2 | 2.23 | 5.18 | 3.26 | 0.10 | [3.07, 3.45] |
| 3 | 1.76 | 8.18 | 3.10 | 0.08 | [2.95, 3.26] |
| 4 | 1.57 | 11.17 | 2.97 | 0.08 | [2.82, 3.12] |
| 5 | 1.56 | 18.14 | 2.98 | 0.07 | [2.85, 3.12] |

For completeness, Figure 4 provides a direct comparison between constant and exponential load distributions.

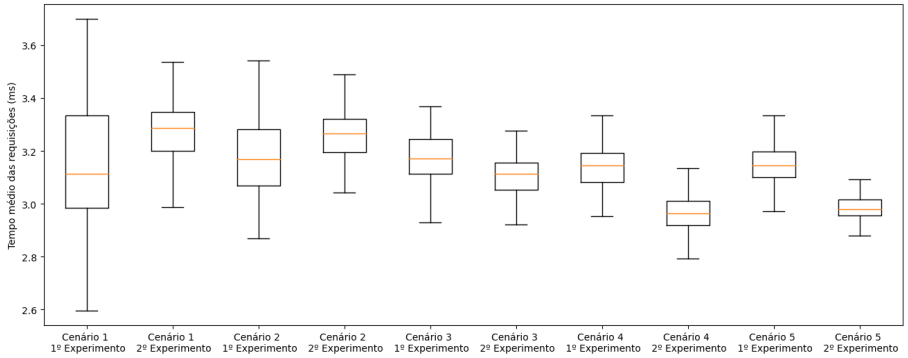


**Figure 4. Comparison of HTTP latency distributions between constant and exponential loads.**

### 5.3. MQTT Message Acceptance Rate

Beyond latency, we evaluate end-to-end reliability through the MQTT message acceptance rate, which captures the proportion of messages successfully processed across the MQTT–Bridge–Kafka pipeline.

Table 7 reports the acceptance rates observed for both constant and exponential load scenarios.

**Table 7. MQTT message acceptance rate under different workloads.**

| **Scenario** | **Constant Load** | **Exponential Load** |
|---|---|---|
| 1 | 54% | 60% |
| 2 | 67% | 61% |
| 3 | 62% | 35% |
| 4 | 56% | 42% |
| 5 | 56% | 47% |

The results reveal that, while latency remains stable, message acceptance decreases under extreme load, particularly in exponential scenarios. This behavior highlights throughput saturation points within the messaging pipeline, offering insights into system limits without compromising responsiveness.

Overall, the results demonstrate that the proposed architecture sustains low-latency operation across a wide range of workloads, while clearly exposing scalability boundaries in terms of message throughput—an expected and measurable characteristic in event-driven IoT systems.

## 6. Discussion

This section discusses the experimental findings in light of the proposed architectural model, focusing on how performance and reliability evolve as the system transitions across workload regimes. Rather than emphasizing absolute performance values, the discussion highlights observed trends, architectural implications, and scalability limits relevant to event-driven IoT systems.

### 6.1. Latency Stability Across Scale Transitions

One of the most consistent observations across both constant and exponential workloads is the stability of HTTP latency. As shown in Figures 2 and 3, median and 95th percentile latencies remain close to 3 ms in all evaluated scenarios. This behavior is further corroborated by the statistical summaries reported in Tables 5 and 6, whose confidence intervals exhibit minimal overlap across scenarios.

From an architectural perspective, this stability indicates that the core request–processing–response path is largely insensitive to increases in request volume within the tested range. This result can be attributed to the asynchronous, event-driven design of the system, where HTTP ingestion is decoupled from downstream processing through messaging layers. As a consequence, transient workload spikes do not immediately propagate back to the request interface, preserving responsiveness. Importantly, this behavior holds even under exponential load growth, a scenario representative of bursty conditions often observed during scale transitions from medium to large deployments. The results suggest that latency is not the primary limiting factor during such transitions.

### 6.2. Throughput Saturation and Message Acceptance

While latency remains stable, the MQTT acceptance rate reveals a different aspect of scalability. As shown in Table 7, acceptance rates decrease under higher workload intensity, particularly in exponential scenarios, where values drop to as low as 35%. This divergence between latency stability and message acceptance highlights a key characteristic of event-driven architectures: scalability limits manifest first as throughput saturation rather than response-time degradation. In the evaluated system, saturation likely occurs at the MQTT–Bridge–Kafka interface, where buffering, partitioning, and I/O constraints become dominant. From the perspective of scale transition, this finding is significant. It indicates that a system may appear stable when evaluated solely through latency metrics, while silently discarding messages under extreme load. Consequently, relying on latency alone would mask critical reliability limitations. The combined analysis of latency distributions and acceptance rates provides a more complete view of system behavior. A limitation of this evaluation is that communication variability typical of real IoT deployments was not explicitly modeled. As a result, latency stability observed in this study reflects controlled infrastructure conditions, and additional variability may arise in wireless or heterogeneous network scenarios.

### 6.3. Implications for Scale-Aware IoT Architecture Design

The experimental results support the notion that transitioning from medium- to large-scale IoT deployments requires careful consideration of throughput management mechanisms. While the proposed architecture sustains low latency through decoupled communication,

message loss emerges as the defining bottleneck at higher scales. This observation reinforces the importance of explicit scale-aware design decisions, such as dynamic Kafka partitioning, adaptive backpressure strategies, and MQTT retry or buffering policies. Without such mechanisms, scale transitions may lead to degraded data completeness even when response-time guarantees are preserved.

Moreover, the results confirm that architectural scalability should be evaluated as a multi-dimensional property. Latency, throughput, and acceptance rates jointly characterize system limits, and improvements in one dimension do not necessarily imply improvements in others.

### 6.4. Lessons Learned from the Scale Transition Perspective

From a broader viewpoint, the findings validate the central premise of this work: scale transition in IoT architectures is not a linear process. As the system moves toward higher workload regimes, different architectural components become limiting factors at different stages. In the evaluated architecture, the transition reveals a clear shift: at lower and moderate loads, the system behaves as a latency-stable pipeline, whereas at higher loads, it becomes throughput-bound. This shift underscores why architectural models must explicitly account for scale evolution rather than assuming uniform behavior across deployment sizes.

Overall, the discussion demonstrates that the proposed event-driven architecture effectively supports scale transitions in terms of responsiveness, while clearly exposing throughput limits that must be addressed when targeting large-scale IoT deployments.

## 7. Threats to Validity

This section discusses the main threats to validity associated with the experimental evaluation and the measures adopted to mitigate them.

**Internal Validity.** The experiments rely on synthetic workloads generated using the *k6* tool and on measurements collected via Prometheus. To reduce instrumentation bias, all scenarios were executed on a controlled private-cloud environment, with homogeneous virtual machines synchronized via `crontab`. Each workload scenario was repeated three times, and the observed variation across runs remained below 5%, reducing the likelihood of random fluctuations influencing the reported results.

**Construct Validity.** Latency and MQTT message acceptance rate were used as proxies for performance and reliability. While these metrics capture key aspects of end-to-end behavior in event-driven IoT systems, they do not encompass other relevant dimensions such as CPU utilization, memory consumption, network overhead, or operational cost. Nevertheless, the selected metrics are consistent with those commonly adopted in the evaluation of distributed messaging systems and IoT platforms.

**External Validity.** The experiments were conducted on a private cloud infrastructure using Ubuntu-based virtual machines with fixed resource allocations. As a result, absolute performance values may differ when deploying the architecture on heterogeneous environments or public cloud platforms. However, the architectural principles, workload patterns, and observed trends are independent of the underlying infrastructure and can be generalized to other deployment contexts. Another limitation is that the experiments

abstract communication heterogeneity typical of practical IoT deployments, such as wireless interference, packet loss, and variable link quality. Therefore, the reported results isolate the behavior of the software and messaging pipeline under controlled infrastructure conditions, and future work should evaluate the architecture under realistic network dynamics.

**Conclusion Validity.** The analysis focused on descriptive statistics, including mean, percentile values, and confidence intervals, without formal hypothesis testing. While this approach is sufficient to identify performance trends and scalability limits, future studies may incorporate formal statistical tests, such as non-parametric hypothesis testing, to strengthen comparative claims between workload scenarios.

## 8. Conclusion and Future Work

This paper investigated the scale transition of event-driven IoT architectures through an experimental evaluation grounded in a medium-scale deployment. By analyzing system behavior under constant and exponential workloads, the study showed that the proposed architecture maintains stable low-latency response times while progressively revealing throughput-related limitations as workload intensity increases. The results demonstrate that scale transition in IoT systems is not characterized by uniform degradation across performance dimensions. Instead, different architectural constraints emerge at different stages of growth. In the evaluated system, latency remains stable across all scenarios, whereas message acceptance rates expose saturation effects in the messaging pipeline under high load. This finding highlights the importance of evaluating scalability using multiple complementary metrics rather than relying solely on response-time measurements. From an architectural standpoint, the study confirms that event-driven designs effectively decouple ingestion from processing, supporting smooth transitions from moderate to higher workloads. At the same time, it emphasizes that explicit throughput management mechanisms are essential when targeting large-scale deployments.

As future work, several directions can be pursued. First, the architecture can be extended with adaptive mechanisms, such as dynamic Kafka partitioning, backpressure-aware consumers, and MQTT retry or buffering policies, to mitigate message loss under extreme load. Second, additional experiments may explore larger-scale scenarios, including geographically distributed deployments and heterogeneous edge devices. Third, incorporating resource-level metrics, such as CPU utilization, memory consumption, and network overhead, would provide deeper insights into performance bottlenecks. Another important extension is the inclusion of formal statistical hypothesis testing to strengthen comparisons between workload regimes. Furthermore, future studies may evaluate the architecture under more realistic IoT communication conditions, including wireless links and variable network quality. Finally, comparative analyses with alternative architectural styles, such as serverless-based event-driven platforms, as well as operational cost and energy efficiency evaluations, represent promising directions for further investigation.

## References

Bonomi, F., Milito, R., and Zhu, J. (2022). Fog computing and its role in the internet of things. *ACM SIGCOMM Computer Communication Review*, 52(1):13–16.

Cugola, G. and Margara, A. (2012). Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3).

Curasma, H. P., Pan, C. F., and Estrella, J. C. (2024). Agents for automatic control of sensors using multi-agent systems and ontologies: A scalable iot architecture. *Procedia Computer Science*, 238:404–411.

da Cruz, M. A. A., Rodrigues, J. J. P. C., Al-Muhtadi, J., Korotaev, V. V., and de Albuquerque, V. H. C. (2018). A reference model for internet of things middleware. *IEEE Internet of Things Journal*, 5(2):871–883.

Daissaoui, A., Boulmakoul, A., Karim, L., and Lbath, A. (2020). Iot and big data analytics for smart buildings: A survey. *Procedia Computer Science*, 170:161–168.

de Assunção, M. D., Veith, A. D. S., and Buyya, R. (2017). Resource elasticity for distributed data stream processing: A survey and future directions. *CoRR*, abs/1709.01363.

Freire, G. M., Curasma, H. P., and Estrella, J. C. (2024). A distributed software architecture for iot: Container orchestration impact and evaluation. *Procedia Computer Science*, 238:224–231.

Ghobaei-Arani, M., Souri, A., and Rahmanian, A. (2020). Resource management approaches in fog computing: a comprehensive review. *J Grid Computing*, 18(1):1–42.

Happ, D., Karowski, N., Menzel, T., Handziski, V., and Wolisz, A. (2017). Meeting iot platform requirements with open pub/sub solutions. *ANNALS OF TELECOMMUNICATIONS*, 72(1-2):41–52.

Khan, L., Yaqoob, I., Tran, N., Kazmi, S., Dang, T., and Hong, C. (2020). Edge-computing-enabled smart cities: A comprehensive survey. *IEEE Internet of Things Journal*, 7(10):10200–10232. cited By 12.

Khan, W. Z., Ahmed, E., Hakak, S., Yaqoob, I., and Ahmed, A. (2019). Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235.

Khriji, S., Benbelgacem, Y., Cheour, R., Houssaini, D. E., and Kanoun, O. (2022). Design and implementation of a cloud-based event-driven architecture for real-time data processing in wireless sensor networks. *JOURNAL OF SUPERCOMPUTING*, 78(3):3374–3401.

Margara, A. and Rabl, T. (2019). Definition of data streams. pages 648–652.

Ngu, A. H., Gutierrez, M., Metsis, V., Nepal, S., and Sheng, Q. Z. (2017). Iot middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal*, 4(1):1–20.

OASIS (2019). *MQTT Version 5.0*.

Varatharaj, M. (2024). Scalable event-driven architectures for real-time data processing: A framework for distributed systems. *International Journal of Computer Engineering and Technology*, 15(6):1952–1965.

Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., and Jue, J. P. (2019). All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330.