From Legacy to Intelligent IIoT Systems: Automation, Scalability and Elasticity

Gianluca Caiazza
Ca' Foscari University
Venice, Italy
gianluca.caiazza@unive.it

Teodors Lisovenko
Ca' Foscari University
Venice, Italy
teodors.lisovenko@unive.it

Pietro Ferrara
Ca' Foscari University
Venice, Italy
pietro.ferrara@unive.it

Fabio Berti
Zamperla
Vicenza, Italy
fabio.berti@zamperla.it

Francesca Ferrari
Zamperla
Vicenza, Italy
francesca.ferrari@zamperla.it

Alessandro Zaupa
Zamperla
Vicenza, Italy
alessandro.zaupa@zamperla.it

Guangzheng Zhang
Zamperla
Vicenza, Italy
guangzheng.zhang@zamperla.it

***Abstract*—The Internet of Things (IoT) revolution is reshaping how physical devices embedded with software connect to the Internet, facilitating seamless data exchange and driving automation. Industrial IoT (IIoT) extends these capabilities to industrial devices and Cyber-Physical Systems (CPS), driving Intelligent Manufacturing. This integration supports advanced applications like remote monitoring, predictive maintenance, machine learning (ML), and artificial intelligence (AI) optimization, enhancing production efficiency, adaptability, and decision-making. However, managing the vast amounts of data generated requires scalable, automated software architectures. Many small and medium-sized enterprises (SME) face challenges in building such systems due to limited resources and expertise, often starting with manual data collection and basic automation.**

This paper presents a solution: a fully automated, configurable, and scalable software architecture for Intelligent Manufacturing. Our system has been operational for almost one and a half years on 21 plants, processing about 17K tasks, amounting to more than three months of computations. The experimental results show that automation and elasticity have been needed by such systems since the beginning, while scalability is not required during an initial experimentation phase.

I. Introduction

As the Industrial Internet of Things (IIoT) continues to evolve, it has garnered significant attention as a natural progression of Distributed Control Systems (DCS) and a cornerstone of Industry 4.0. Achieving IIoT-driven modernization typically revolves around three core principles: connectivity, data acquisition, and analysis [1]. These foundational elements are essential for transforming traditional industrial operations into intelligent, data-driven ecosystems, as highlighted by advancements in technological readiness [2], [3].

However, transitioning from legacy industrial systems to IIoT presents numerous challenges. Initial efforts often focus on demonstrating feasibility through Minimum Viable Products (MVPs) and prototypes, which are crucial for validating concepts and gaining stakeholder buy-in [4]–[6]. Scientific endeavors have emphasized automation, scalability, and elasticity as key objectives for IIoT systems, yet significant gaps remain in practical application. For instance, certain regions report limited automation knowledge despite strong motivation for progress [7], while integration with cutting-edge technologies such as machine learning and AI-driven analytics introduces further complexity [8].

The existing literature addresses specific facets of IIoT—such as network optimization [9], theoretical automation models [10], and frameworks like Lean and Six Sigma [11]—but often lacks actionable solutions for industries with established infrastructures. Particularly in brownfield scenarios, where legacy systems must coexist with modern advancements, practical and scalable approaches remain scarce. Most "Roadmaps to Industry 4.0" focus on automation and scalability [12] but are often ill-suited for industries that already operate functional IIoT architectures.

This paper seeks to bridge this gap by presenting a practical approach to extending IIoT capabilities in a brownfield industrial context. We introduce a Custom Workflow Engine (CWE) designed to integrate seamlessly with legacy architectures while enabling scalability, resource efficiency, and support for dynamic workloads. Through a case study of a brownfield industrial plant, we demonstrate how this approach addresses current challenges and lays the groundwork for resilient and adaptable industrial systems.

Related work Industries are often inherently conservative, with their infrastructure deeply rooted in legacy systems or on the brink of being classified as such. This cautious stance arises primarily from the substantial costs, risks, and operational disruptions associated with replacing established infrastructures. Consequently, software in these environments is typically highly specialized, reflecting years of accumulated domain expertise, and is often proprietary and closed-source. These systems necessitate finely tuned operational environments, which can be challenging to modernize without disrupting their core functionality.

Containerization has emerged as a transformative solution in this landscape, offering the portability, flexibility, and isolation required to support and evolve these specialized systems. By encapsulating applications and their dependencies, containerization ensures consistent performance across diverse environments and provides a bridge between legacy systems and modern infrastructure demands. This approach facilitates innovation while maintaining stability, allowing industries to adapt incrementally to technological advancements.

Nearly a decade ago, Goldschmidt and Hauck-Stattelmann [13] highlighted Docker's ability to address flexibility, distributed architecture, and legacy system preservation, emphasizing its role in meeting the demands of modern industrial systems. More recently, Di Modica and Foschini [14] surveyed lightweight virtualization in Industry 4.0 manufacturing, recognizing Docker as a frequently used technology, often with microservices acting as functional elements within containers. Similarly, Basem [15] concluded that "container-based solutions are the basis for future automation" in industries, requiring technologies like Kubernetes.

While Docker has emerged as the de facto standard for containerization and Kubernetes as the leading orchestration engine, architects today have a range of alternative options, variations, and combinations to consider beyond these two. Kumar et al. [16] provide a comparative analysis, clearly outlining that, for each layer—whether infrastructure (e.g., cloud or private on-site data center), operating system (e.g., Windows or Linux), container runtime (e.g., Docker runtime, CRI-O, or Containerd), container technology (e.g., Docker or LXC), or orchestration engine (e.g., Kubernetes, Docker Swarm, Apache Mesos)—multiple choices exist. The selection of the appropriate stack must be guided by the available features, priorities, and case-specific requirements.

The work of Alamoush and Eichelberger [17] identifies 23 Industry 4.0 requirements related to orchestration, containerization, and system-level concerns, accompanied by a detailed feature taxonomy. Of particular relevance to our work are the system-level requirements *SysR-2 (enable scalability)* and *SysR-5 (provide flexibility and extensibility)*, in addition to the *license* feature emphasizing the use of free and open-source solutions. Other features, such as AMQP protocol support, are also crucial to our mission for automation, as they ensure reliability, which will be further elaborated in Section IV. Additionally, the inclusion of on-premise deployment, which supports locally managed infrastructure and safeguards data privacy, aligns with our focus. In light of these considerations, cloud-based solutions such as Amazon Web Services (including Fargate, Lambda, and EventBridge), Google Cloud Run, and Azure Kubernetes Services are outside the scope of our study due to their inherent limitations in flexibility and potential vendor lock-in. Platforms such as Red Hat OpenShift, while capable of on-premises deployment, also require licensing, thus complicating their use for our purposes. Solutions such as HashiCorp Nomad and Docker Swarm were deemed unsuitable due to their lack of clear event-based support for scalable solutions (both native or via extension), requiring custom scaling approaches. Likewise, the absence of AMQP protocol support disqualified Apache Mesos from consideration. Alamoush and Eichelberger conclude that Kubernetes offers the broadest set of features, positioning it as a "Swiss knife" for container orchestration [17].

While robust, Kubernetes does not inherently address challenges associated with dynamic scalability and automation. Built-in mechanisms like the Horizontal Pod Autoscaler (HPA) dynamically adjust resource allocation based on CPU and memory usage but may fall short in event-driven scenarios. As we detail in this paper, our system faces high-frequency, triggered events that standard solutions are ill-equipped to handle, which is why Kubernetes was selected for its extensibility.

Addressing similar issues, Mukherjee *et al.* [18] integrated KEDA to reduce application latency in time-sensitive contexts, where resource-consumption-based scaling proved inadequate. Although their focus on replica placement across distributed servers diverges from our objectives, it presents a compelling opportunity to explore hybrid deployments mixing on-premises and cloud resources.

Other studies have adopted simplified yet effective strategies. Wang *et al.* [19] utilized KEDA to reduce architectural complexity through a model-driven approach, while Korontanis *et al.* [20] scaled systems using RabbitMQ's internal message queue length, achieving higher scalability. Figueira and Coutinho [21] demonstrated cost-effective self-adaptive microservices leveraging KEDA, particularly where HPA alone struggled with scaling inefficiencies.

These findings led us to select Kubernetes in the drafting phase of our system as the most supported, feature-rich, and mature option, which can be extended with tools such as KEDA. This choice allows us to build a serverless, on-premises solution that includes privacy safeguards and can be ported to countries with specific physical server requirements. Additionally, Kubernetes offers the flexibility to support various container technologies, enabling future replacement of conventional Docker with alternatives such as Containerd or CRI-O. Choosing Kubernetes allows for migration to lightweight distributions like MicroK8s or K3s in future development cycles.

Contribution To the best of our knowledge, no prior work has comprehensively addressed migrating a legacy brownfield IIoT system to an automated, scalable, elastic, agnostic architecture capable of handling dynamic workloads. Most literature focuses on consolidating third-party cloud solutions. It emphasizes cloud- or fog-based approaches, often overlooking the implications of maximizing resource and energy efficiency as discussed in Sections IV and V.

Our work bridges this gap by introducing an agnostic, adaptive system capable of supporting diverse, sudden workloads across the entire IIoT ecosystem. By prioritizing scalability and elasticity, we establish a robust framework that addresses immediate industrial needs while aligning with long-term sustainability and operational resilience goals. This contribution marks a step forward in enabling intelligent, resilient, and environmentally conscious IIoT systems, paving the way for the next generation of industrial automation.

The paper is structured as follows. Section II introduces background about IIoT. Sections III, IV and V discusses the initial architecture, how it was automatized, and then made scalable and elastic. The Section VI reports the experimental results, while Section VII concludes.

II. Background

In this section, we introduce the main concepts and technologies that motivate the work we present in this paper. For the sake of simplicity, we oversimplify most of the concepts (that alone might require an entire article to be described), focusing only on what is relevant to our approach.

A. Programmable Logic Controller

Starting from the late 60s, the industrial world experienced the advent and rise of industrial computers called Programmable Logic Controllers (PLC) [22]. The primary purpose of these computers is to control the manufacturing processes by interacting with various devices involved in manufacturing. At an extremely high level, these computers are wired to the devices involved in the manufacturing process, and they interact with them by reading some data, usually coming from sensors, and taking some actions through actuators.

Born in the late 60s for the automotive industry, PLCs quickly gained popularity since they allowed to supervise and automatize several industrial processes that otherwise might be less reliable and would require human intervention. They are pervasive in the industry nowadays, and all modern industrial plants contain some PLCs in some forms or others. While their technology is highly complex and has more than half a century of evolution, for this paper, we oversimplify them as industrial computers that might read from and write data to industrial plants. PLCs are usually programmed by professionals who are engineers with deep knowledge about the manufacturing process domain but no programming expertise. For this reason, PLCs can usually be programmed with graphical diagrams (e.g., IEC 61131-3 standard).

B. Industrial IoT

A recent trend that appeared about a couple of decades ago in another field is the so-called Internet of Things (IoT) [23]. Another primary trend that appeared in parallel with PLCs was embedded software. During the 60s, more and more physical electronic devices started to be equipped with some software (called embedded) that is executed on the physical device. Because of this, this type of software is highly specialized on some hardware with various constraints.

IoT represents the natural evolution of embedded software. Thanks to the advent of public and local networks, various *things* (usually equipped with embedded software) got connected through some networks. This technological shift allows things to communicate among themselves, to be locally coordinated by some other components (e.g., edge nodes), and to be remotely accessed and controlled. Like PLCs, we oversimplify the IoT paradigm with the ability to access the data produced by things remotely and to take their control.

Then IoT naturally extended to Industrial IoT (IIoT) [24], that is, IoT's simple and plain application to the industrial world. In such a context, PLCs are usually the so-called things of IoT architectures. Therefore, the distance between PLCs (or other industrial devices) and everyday *things* makes these two concepts radically different from many perspectives. For what concerns this paper, we focus our attention on the fact that IIoT allows (i) access to data coming from and (ii) remote interaction with industrial plants. The first point can be used to visualize and collect data that can be applied to various business-critical tasks, such as preemptive maintenance and optimization of the manufacturing process. The second point allows for remote supervision of the manufacturing process, opening the door to tasks like manual or automatic maintenance and optimization.

Small and medium-sized industries usually do not have teams specialized in software development and engineering since most of the programming of PLCs does not require high specialization in software programming. Therefore, industrial organizations usually approach IIoT by professionals with deep expertise in the manufacturing process but limited software development skills.

C. Industrial Big Data

Moving one step further, IoT was an enabling technology for the appearance and development of big data [25]. Big data represents the general concept that vast quantities of data are available for inspection, analysis, etc. Collecting, storing, analyzing, and visualizing such data is always challenging due to data representation and structure heterogeneity. Things produce data through their sensors, and IoT enables such data collection. All these considerations extend to IIoT naturally, where the quantity of sensors is usually in the order of magnitudes higher, and the collected data has a significant business value. In particular, industrial big data [26] comprehends a massive variety of data coming from different sources, where IIoT plays a key role. The development of intelligent systems based on artificial intelligence requires big data, and in an industrial context, it led to the development of intelligent manufacturing. Therefore, collecting and processing industrial big data is an enabling technology for intelligent manufacturing.

However, deep software development, engineering, and architecture expertise are required to utilize this data effectively. In particular, industrial big data requires IT systems that can automatically treat this data and scale up to vast quantities of data. This context requires developing, deploying, and maintaining complex software architectures utilizing modern technologies. As discussed in this section, this is outside the competencies usually in place inside most existing SME manufacturing industries.

What usually happens in practice is that engineers with no particular programming skill start to develop various systems to understand and exploit the possibilities of intelligent manufacturing. Such a process always combines various manual steps and the execution of various scripts developed to semi-automatize simple tasks. If, on the one hand, this work is needed to understand the value of intelligent manufacturing and convince managers internally to allocate resources to the development of such a system, on the other hand, it produces a legacy system even before the system is fully developed. The scripts contain a deep knowledge about the manufacturing system and the data it produces that can hardly be reproduced by expert software developers without a specific knowledge of the specific industrial system.

D. *Software Architectures*

Software architectures have been defined, developed, and executed for several decades. The experience gained by this scientific community and industrial applications is precious, and it already provides several solutions to automatize and scale computations. Architectural paradigms such as Service-Oriented Architecture (SOA), microservices, and serverless computing provide a solid foundation for designing such systems. Containerization is the primary means of accommodating a wide range of technologies, vendors, and specific environments across various industrial segments. It serves as the foundational architectural unit, scalable through orchestration solutions. Synchronous communications through the HTTP protocol and REST APIs and asynchronous communications through the publish-subscribe pattern are the standard ways that allow different software architecture components to communicate. In our work, we take benefit and apply all these concepts and technologies to create an automatic and scalable software architecture for managing industrial big data.

III. Legacy IIoT Architecture

This section introduces a legacy industrial architecture developed by a leading amusement ride manufacturer, which integrates essential monitoring and control functionalities for accessing multiple remote sites (parks), each containing one or more rides. In this context, a "ride" refers to a complex industrial system subjected to significant mechanical and operational stress, typically controlled by a PLC. Continuous and effective monitoring is critical for maintenance, enabling the detection of anomalies such as unusual vibrations, signs of corrosion, or wear and tear that could threaten the ride's structural integrity. Furthermore, parameters such as fatigue, speed, acceleration, braking performance, and launch mechanisms must be precisely monitored to ensure the ride operates safely. Although initially not designed with data collection principles in mind, a ride incorporates several key elements vital for IIoT integration, such as sensor data access, real-time data polling, and operational feedback loops. By leveraging PLC programming, we can access, collect, and process sensor data, providing real-time feedback by monitoring critical parameters, ensuring safe, efficient, and reliable operations across the entire ride's duty life cycle.

A. *Existing Architecture*

Fig. 1 depicts the dataflow architecture. Beginning at the highest level of the diagram, parks are distributed globally, each hosting one or more rides. Each ride has actuators and sensors that control and monitor critical parameters such as speed, temperature, energy consumption, etc. The sensors

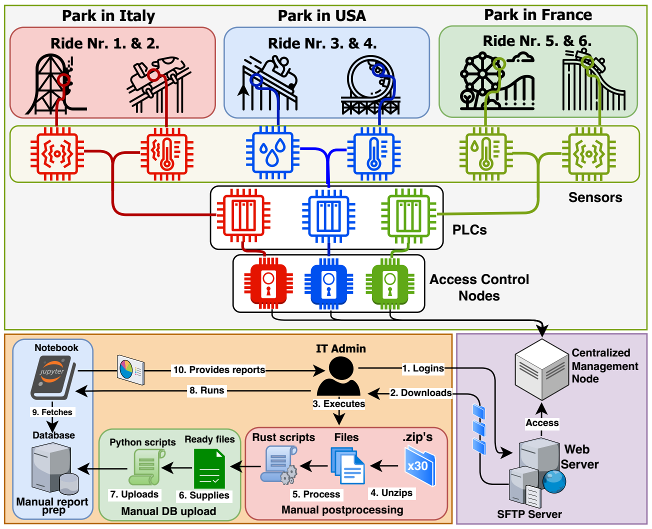

Fig. 1: Legacy Architecture for Reporting

collect real-time data, which is then transmitted to a PLC. The PLC processes and converts this data into a digital format, making it accessible for subsequent analysis and monitoring through general-purpose computing devices.

1) Things and Connectivity: PLCs are designed for localized, on-site control and are not intended to be connected to the Internet. These devices operate within trusted networks, where the integrity and security of communication are assumed and, therefore, are not built with cybersecurity measures. As trusted components, PLCs are optimized for executing specific industrial tasks with high reliability. Extending their functionality beyond their intended purpose would compromise their performance in hard real-time computation, which is critical for ensuring operational safety.

To address these limitations and protect the PLC from potential threats or overloading, a local interfacing layer—a computer commonly referred to as an IoT edge—is introduced at the Access Control Nodes' level. This IoT edge device serves multiple purposes: it facilitates data collection and storage, securely transmits data to external systems, and acts as a protective gateway, preventing any direct connection between the PLC and external networks. By isolating the PLC from direct Internet exposure, the IoT edge ensures that the PLC remains dedicated to its core functions while safeguarding it against potential cyber threats and unauthorized access.

As a result, we leverage these devices to securely connect to a centralized web server, facilitating communication with the IoT edges. These IoT edges, connected to the PLCs, collect all the data produced by them. As illustrated in Fig. 1, the process begins with authentication (Step 1) to the web server, manually performed by a human operator. The server provides an SFTP (Secure File Transfer Protocol) tunnel, enabling secure data retrieval and processing to and from the IoT edges.

2) Manual Data Workflow: In the described system, sensor data is collected daily and compressed into multiple ZIP archives. This method optimizes storage and transfer efficiency by minimizing the data load for the IoT edge, facilitating faster and more manageable data handling. These compressed archives are then downloaded into the centralized server (Step 2), manually extracted (Steps 3,4), and processed (Step 5). Then the central post-processing phase begins, where external scripts (Step 6) are used to prepare the data for upload to the database. This phase involves critical tasks such as data sanitization, formatting, and structuring, all performed manually by the operator. This ensures that the data is cleaned, organized, and aligned with the system's specific requirements, with further modifications made manually if necessary.

It is important to note that these scripts represent a legacy artifact within the IIoT system. Despite their essential role in enabling core data processing functionalities, the workflow remains constrained by human involvement, requiring significant time from operators that could be allocated to other tasks. While practical, this human dependency highlights an area for optimization in terms of automation, which could streamline the overall process and reduce the operational burden on IT personnel, as discussed in Section III-B.

Following the post-processing stage, we focus on the data pipeline, uploading the prepared data. This phase is managed by several Python scripts, which receive the processed files from the previous steps (Step 6). The role of these scripts is to facilitate the structured upload of data to the manufacturer's centralized database (Step 7). Once the data is stored, it can be queried for further analysis or reporting.

The final phase of the process (Step 8) involves the IT personnel's manual execution of the Reports' script (generally Jupyter Notebook or Matlab). These scripts are powerful tools for generating detailed reports, including visualizations such as plotted graphs and diagrams based on the processed data. We retrieve the necessary data from the database (Step 9) and produce the required visual outputs to generate these reports. Once the data is retrieved and the analysis is completed, the generated reports (Step 10) are delivered in some predefined formats to facilitate sharing and future reference. The overall workflow, as outlined, enables the systematic collection, processing, and storage of sensory data, ensuring that it is readily accessible for further analysis.

B. Limitations and Areas for Improvement

While the current system successfully demonstrates core functionalities, it also reveals several limitations that must be addressed for optimal performance. This section thoroughly explores these constraints and outlines potential avenues for improvement, which we have leveraged to enhance system performance, scalability, and elasticity. These aspects are further explored with the prototype introduced in Section IV, while scalability and elasticity improvements are discussed in the context of the system's evolution in Section V.

1) Manual-Driven System: The current system's critical limitation is its reliance on manual processes, as illustrated in Fig. 1. Every stage of the system, from logging in to extracting files from archives, is human-driven. These tasks, which are routine and repetitive by nature, consume substantial time and resources. This inefficiency hinders overall productivity and diverts attention from more value-added activities, such as analyzing and interpreting the data. One of the fundamental goals of *Industry 4.0* is to minimize human involvement in such low-level operations, enabling workers to focus on higher-order tasks [27], particularly those related to analytics, control, and strategic decision-making. The absence of automation in this context undermines the system's potential for optimization and scalability, highlighting a key area for improvement.

2) Interoperability Gaps and Technical Debt: The technical debt accumulated from relying on various programming languages and frameworks is a significant limitation of the system's architecture. While these choices were made to address immediate functional needs, they resulted in system fragmentation. This fragmentation impedes the potential for seamless interoperability, restricting the efficient communication between components across different platforms. Although delegating different workflow phases to various programming languages is not inherently problematic, creating a coherent, unified system to orchestrate these steps presents a significant challenge. Integrating the core executables into a cohesive system is far from straightforward for the legacy system discussed in this section. While a human operator can interpret the various outputs from scripts and executables—often written by various authors with inconsistent input/output structures—this becomes a complex task for an autonomous orchestrator, which must navigate these inconsistencies. As a result, this technical debt continues to grow, making it increasingly difficult to maintain and evolve the system without substantial rework. To address this, we tackled the challenge by decomposing the system into microservices and establishing standardized communication protocols, thus laying the groundwork for more efficient orchestration and future scalability, as discussed later in this paper.

3) A Scalability and Elasticity Bottleneck: A critical challenge identified during testing was the inability to connect to parks and download data while operational, which posed a risk of overloading the network. This issue would be manageable if only one park operated within the same timezone, allowing manual retrieval outside working hours. However, the complexity increases significantly when managing multiple parks across the world. For parks on the opposite side of the globe, night personnel must perform routine data retrieval tasks, which introduces logistical inefficiencies. Such an approach is not sustainable in a manual context, emphasizing the need for a programmatically scheduled solution integrated into the system to automate these operations efficiently.

4) Identifying Opportunities for Improvement: While scalability and automation are primary motivations, our ultimate goal is to incorporate the foundational properties of Intelligent Manufacturing. Legacy architectures, like the one presented here, often operate as one-way systems, where functionality is limited to data visualization without the capacity for proactive actions, such as triggering alerts or dynamically adjusting system parameters. Modernization efforts aim not only to improve maintenance processes but also to enable predictive and prescriptive capabilities, allowing systems to forecast potential failures and autonomously act to mitigate them.

This transformation requires integrating key attributes such as interoperability—to facilitate seamless communication between disparate components—and connectivity, ensuring continuous data exchange via IoT technologies. Additionally, introducing real-time monitoring and control enhances system responsiveness, while data-driven decision-making leverages analytics to optimize workflows and outcomes.

IV. Automation

To address the constraints identified in the previous Section, we systematically broke down the workflow into manageable, minimal steps. By leveraging containerization with Docker, we transformed the architecture into a modular system that simplifies deployment, enhances maintainability, and provides the flexibility necessary for future development. A crucial aspect of this process was distinguishing between synchronous and asynchronous components, which are fundamental in ensuring the system's efficiency and responsiveness. In this Section, we introduce the implementation of an intermediary architecture, referred to as the Containerized Workflow Engine (CWE), depicted in Fig. 2. This architecture serves as a bridge, transforming the rigid workflows of the legacy system into an adaptable and automated framework. The discussion is structured into three essential parts: (i) we address the structural separation of the system by identifying and splitting its components based on logical separation of concerns and execution contexts, (ii) we examine the procedural dependencies among the various workflow components by focusing on how they can be coordinated to operate autonomously, (iii) finally, we discuss the role of human operators, detailing how and when their involvement is required. This ensures that while the system is largely automated, it retains flexibility for operator intervention when necessary.

1) Containerized Architecture: Depicted in the lower region of Fig. 1 are the distinct action blocks: download, post-processing, upload, and report generation. While these blocks offer a clear conceptual workflow, their execution relies on monolithic scripts as described in Section III. To avoid dependency conflicts, such as compatibility issues or constraints tied to specific versions of interpreters and compilers, we adopted a modular approach by isolating the system's components based on their logical roles and execution contexts. For example, the post-processing phase is delegated to a container running a *Rust* executable, while legacy scripts and database uploads are managed by containers utilizing Python.

Containerization through *Docker* forms the backbone of this architecture, transforming the system into an ensemble of modular, self-contained units. Each container operates within its own execution context, recreating a new process per iteration of work. This ensures a clean and isolated workspace, eliminating the risk of issues from corrupted execution from other modules or faulty processes. Additionally, this containerized architecture allows for the migration of components, enabling different servers to migrate or distribute workloads. As a result, if the container tasked with weekly report generation encounters an error, the system does not revert to redundant processing steps like re-downloading or re-processing completed phases. Instead, a new report process is initiated to resume solely from the reporting phase without affecting other elements. This modular and isolated architecture is encapsulated within the *data component* depicted in Fig. 2, effectively realized through a *Docker* composition that we analyze in the following in further detail.

2) Autonomous Operations and Workflow Coordination: In the following, we discuss procedural coordination, focusing on how components can be orchestrated to operate autonomously, minimizing human interaction while ensuring robust performance. A principal challenge involves determining how to replace manually performed actions while defining what tasks should remain under human oversight. Considering the operational constraints of managing tens of parks and their specific time windows for data accessibility, the core actor in the CWE architecture is the scheduler, implemented using the *Python* library APScheduler (https://apscheduler.readthedocs.io/en/3.x/). As part of the *data component*, the scheduler operates as a standalone unit, packaged alongside a Flask web application that exposes an endpoint serving as the container's entry point. This endpoint processes incoming HTTP requests, formulated as cron job definitions, to trigger scheduled tasks. While the scheduler runs synchronously to handle task initiation, extending this synchronous approach further would be inefficient. Delays in processes, such as late data downloads or unexpectedly long post-processing times, could lead to cascading issues. For instance, if a new trigger arrives while the previous task is still ongoing, it must wait until the preceding operation completes. Moreover, a late-triggered process might initiate outside the acceptable time window for file downloads, potentially overloading the park's network. To address these challenges, the architecture must divide tasks into separate containers and operate asynchronously. With three independent actors (i.e., downloader, post-processor, and report), robust supervision is essential to ensure that each unit operates sequentially and without conflicts. Post-processing, for example, should only commence once the downloader unit has successfully completed its task and reported its status. To achieve this, the *orchestrator* instance is introduced as a central coordination mechanism, where each instance of the orchestrator is dedicated to managing a specific pipeline execution. When the scheduler triggers a new task—downloading, post-processing, reporting, or the entire pipeline—it spawns an instance in the orchestrator tailored to that particular request. The orchestrator's processes operate synchronously, determining whether workflows can be executed in parallel or must be run sequentially, and ensuring that each can be stopped individually without impacting others. This design allows for flexible coordination, particularly when simultaneous operations or interruption requests must be handled efficiently. Each orchestrator instance keeps track of its assigned workflow, ensuring every step is completed in sequence before progressing to the next phase. For example, the orchestrator initiates the

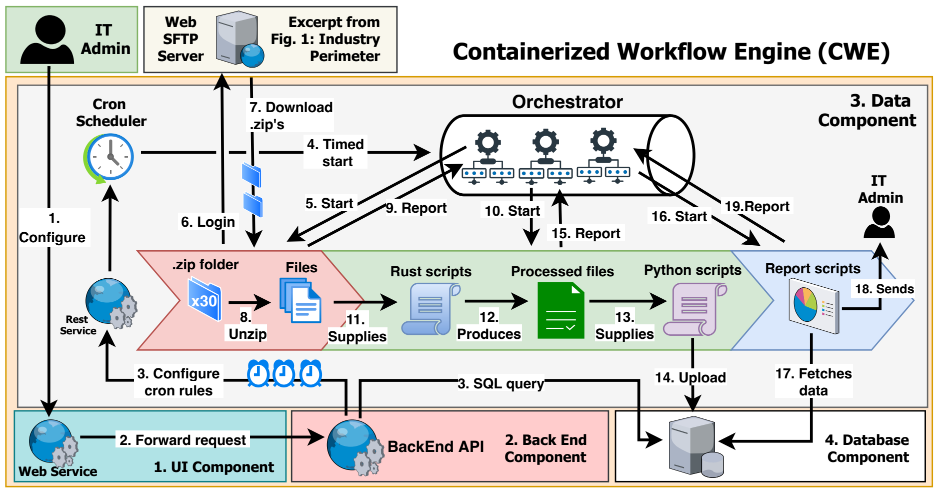

Fig. 2: Architecture for Automated Reporting: Containerized Workflow for Downloading, Post-Processing, and Reporting

downloader process, monitors its execution, and waits for a success message before triggering the post-processing unit. The same applies to report generation, ensuring an orderly and synchronized flow through the pipeline. In addition to coordinating tasks, the orchestrators are designed to handle errors. Each orchestrator logs any encountered issues and identifies the specific pipeline stage where the error arose. If repeated failures persist, the orchestrator escalates the situation by notifying a human operator for manual intervention.

3) From Operating the Gears to Overseeing the System: With human operators freed from the repetitive tasks of initiating and managing workflows, their role within the IoT reference model is elevated to the upper layers, emphasizing strategic oversight and decision-making. The system requires a user-friendly application interface to support this shift for configuring schedules and querying system status. We introduce two additional components within the CWE architecture to meet this need. The *UI component* is a dedicated container that delivers an intuitive web-based graphical interface, enabling IT personnel to perform system configurations, such as adding parks and rides or adjusting operational parameters. Built as a *Flask* application in *Python*, this component operates as a frontend, forwarding user interactions and configuration requests to the backend for processing. The *backend component*, another *Python*-based web API, serves as a dispatcher and intermediary, facilitating communication between the user-facing interface and the supervised system (i.e., data component), which ultimately orchestrates the underlying structure. It handles HTTP requests for the cron job scheduling, ensuring seamless coordination between user commands and system execution. Additionally, it processes more simple queries, such as retrieving job failure logs. Lastly, the database, illustrated in Fig. 2, is the central repository for post-processed data and facilitates report generation. While the primary goal of the CWE is to automate workflows while preserving legacy functionality, the system's design achieves several secondary benefits. By abstracting the complexity of workflow management, personnel with minimal knowledge of the system's internal workings can now interact effortlessly with the application. Each architectural component resides within its dedicated environment, embodying the principles of microservices. This modularity ensures that each part remains isolated, extensible, and independent of the others. Another achievement is maintaining control over data privacy and security. Managing all generated data within the system's architecture mitigates the risk of exposing sensitive information to external platforms.

V. Scalability and Elasticity

The architecture of Section VI demonstrated both functionality and efficiency. However, soon inherent limitations became evident. Specifically, with an increase in incoming requests, the Dockerized "workers" faced resource constraints tied to the finite capabilities of their host servers. Additionally, the orchestrators' throughput introduced a critical bottleneck. In this Section, we introduce an enhanced architecture that evolves from a server-centric design to a Kubernetes-based approach to address these challenges. This transition harnesses the asynchronous nature of the system's workflows, optimizing resource utilization by provisioning only the resources strictly necessary at any given time, thus enabling full exploitation of the available computational power.

1) Restructuring into Execution Pods: The previously Dockerized components have been restructured as Pods managed as ScaledJobs in Kubernetes. Each Pod is allocated a predefined range of minimum and maximum resource limits, dynamically adjusted by Kubernetes based on real-time demand. This restructuring ensures that resource allocation scales up and down in proportion to the workload, allowing architectural scalability and elasticity. Additionally, each Pod operates within a completely independent execution environment. While this design introduces initialization overhead for each task—resulting in some resource inefficiency due to the use of identical pods—it remains advantageous as it ensures true job independence and leverages the elasticity and scalability of the cluster for concurrent task execution.

2) Transition to Event-Driven Decentralized Scaling: The architecture adopts an event-driven scaling model with a decentralized approach, leveraging priority-based queues managed through RabbitMQ and the Kubernetes Event-Driven Autoscaler (KEDA). This design dynamically adapts system components based on workload demands, as illustrated in Fig. 3. RabbitMQ is a more reliable, high-throughput queueing system that replaces the centralized orchestrator, ensuring improved message management and task scheduling. By scaling resources only when required, the architecture achieves resource efficiency and reduces energy consumption by shutting down unused components during periods of inactivity.

3) Priority-Based Queues and Decentralized Coordination: KEDA, in conjunction with RabbitMQ, ensures precise scaling of system resources. RabbitMQ manages communication through multiple prioritized queues, enabling fine-grained control over resource allocation. High-priority tasks are handled promptly, ensuring that critical workflows are never delayed by less urgent processes (e.g., downloaders have the highest priority and reports the lowest). Unlike the original architecture, which relied on a centralized orchestrator, the orchestration logic is now delegated to trigger-based cascade rules that communicate via RabbitMQ's jobs handling primitives (omitted for simplicity in Fig. 3). This shift eliminates the bottlenecks associated with centralized coordination and significantly improves fault tolerance. If a single Pod encounters an error, it can be restarted independently. RabbitMQ further enhances system reliability by implementing preconfigured retry thresholds for failed tasks. If failures persist, the system escalates the issue to a human operator, supplying detailed logs to expedite troubleshooting and resolution.

4) Enhanced Monitoring and Observability: The architecture incorporates Kubernetes-native monitoring tools to ensure system reliability and provide actionable insights. Prometheus collects and aggregates key metrics, such as resource utilization and task completion times, while Fluentbit facilitates centralized logging for comprehensive performance tracking. These tools empower operators with real-time visibility into the system's health and performance, enabling proactive maintenance and quick intervention when needed.

5) Advantages of the Kubernetes Model: Adopting event-driven scaling and decentralized coordination effectively eliminates single points of failure, resulting in a more resilient and fault-tolerant system. By dynamically optimizing resource utilization, we minimize operational costs and ensure sustained high performance. Furthermore, integrating advanced monitoring tools significantly enhances system observability, providing real-time insights that enable the architecture to remain elastic and scalable, well-positioned to meet evolving demands and future growth.

6) Architecture as a conceptual template: While certain aspects of the system are case-specific—such as the three-step workflow, containerized scripts, and the goal of report generation and delivery—the proposed architecture serves as a transferable template for other scenarios. It accommodates typical stages of data acquisition, processing, and output while maintaining resource efficiency through its serverless approach. This design is scalable for both small and large datasets, ensures privacy through on-premises deployment, and operates automatically.

The technological stack is modular and flexible, allowing components to be adapted or replaced without compromising functionality. For instance, while Docker is our containerization choice, it can be substituted with LXC on Linux handled environments or Kata Containers for enhanced security. Similarly, RabbitMQ as the message broker can be replaced with alternatives more suited to specific requirements.

In our case, event-driven autoscaling via KEDA is essential, but regular HPA may suffice in simpler scenarios, or Knative could be used for HTTP-based scaling. The architecture is intentionally designed to remain agnostic and adaptable, handling typical IIoT workflows while supporting modifications. Monitoring is decoupled and integrated via agents like Prometheus, ensuring flexibility for evolving technological needs.

VI. Experimental Results

The CWE software architecture started to be operative on May 24th, 2023. We downloaded all the system logs on September 23rd, 2024, and therefore, we discuss the results of 1 year and 4 months of operations here. The IIoT system was adopted in 27 distinct parks in 9 UTC time zones and 39 rides. Six parks had 0 rides (parks 12, 13, 18, 21, 24, and 25). Therefore, from now on, we will consider only 21 parks as the baseline. We anonymized the data by (i) removing the name of the park, (ii) locating it with just the delta of the time zone w.r.t. UTC instead of the name of the timezone, and (iii) removing the name of the rides[1]. We believe these details do not affect the discussion of our experimental results.

We started by inspecting how many rides were tracked per park. Only six parks (parks 0, 5, 6, 10, 4 and 1) have more than one ride related to them, while all the remaining parks have only one ride in the system. The main reasons behind that are that (i) the enterprise producing the rides has only a few (if not one) ride in a single park (since parks comprise rides produced by different companies), and (ii) the system is still under evaluation in most of the parks because of its recent conception, and therefore it is tested on single rides. However, as we will show in this Section, even this number of rides justifies the need for an automated, scalable, and elastic software architecture. It is worth noticing that overall, the enterprise that manages these rides produces around 150 rides per year, and the lifetime of rides is, on average, 20 years. This means that around three thousand rides are currently active, representing the maximum number of devices that might be connected to the system. While it is reasonable to assume that only a relatively small percentage of rides will adopt the

[1]Note to the reviewers: this anonymization is due to the constraints imposed by the enterprise owner of the IIoT system and not to double-blind anonymization.

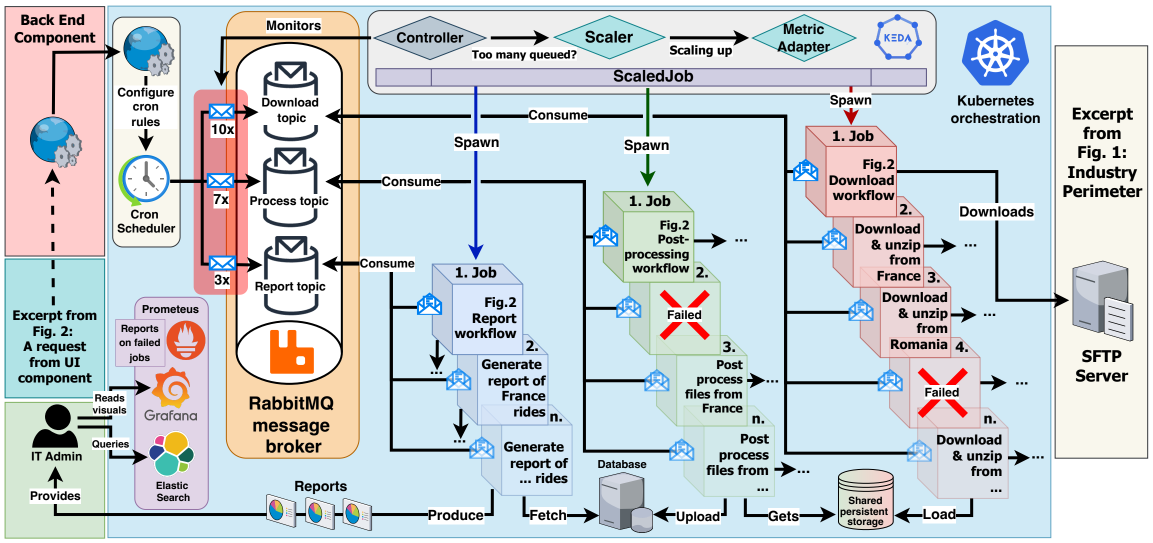

Fig. 3: Scaled IIoT infrastructure

system, the expectation is that several hundred rides will be connected once the architecture is completely activated and added to the machines.

Our experimental evaluation focuses on two main research questions that reflect the contributions of our architecture:

RQ1 Is an automatized architecture an enabling feature for innovative manufacturing?

RQ2 Is scalability and elasticity required by an innovative manufacturing environment?

All the experiments were conducted on a Virtual Machine on VMWare 11.3.0 with an 8 CPU Intel Xeon Gold 5217, 62GB of RAM, and Ubuntu 20.04.6 LTS. The edge node collects values for each sensor every 50 milliseconds on around 200 sensors on average, storing data in JSON format and compressing it, producing around 50MB of compressed data daily per ride on average.

A. *RQ1: Overall computation*

In order to answer the first research question, we are going to study the overall computational time.

First, let us define a baseline comparison with the legacy approach, where a single user had to manually perform several tasks as described in Section III. In such a context, this leads to (i) download data required to connect to the ride via SFTP manually, start the downloading, wait for the end, and disconnect; (ii) postprocess the data required to execute two scripts, waiting for their termination manually; and (iii) produce the reports required to start the computation of the Python notebooks and wait until their termination.

Note that small and medium enterprises, as described in Section II-B, usually have a very small IT team. In our case, the team was composed of four people. Overall, they could dedicate a few monthly hours to the legacy system. Therefore, the legacy system was mostly adopted on demand to produce reports on specific rides when asked by their owners or maintainers or during fairs and exhibitions of this specific industrial sector. After the deployment of the architecture described in this paper, the team started using this online system both for demo purposes and to monitor some rides daily. When collecting the computational times, we divided the tasks between Download, Postprocessing (comprising unzipping and uploading the data to the database), and Report.

Type	#	Overall time	Time per task
Downloader	6490	61 days 12h53’	13’39”
Postprocessor	8252	34 days 11h42’	6’02”
Report	2123	1 day 09h53’	57”

TABLE I: Overall computational times

Table I reports the computational times divided per task. Overall, the computational times sum up to about 97 days and 10 hours of computation (over the 16 months monitored), to more than 2300 hours. This is equivalent to about one and a half years of a full-time employee. Looking into the times, about two-thirds of the time was spent in downloading, about a third in postprocessing, and less than 2% on reporting. On average, a single downloader task requires more than 13 minutes, a postprocessor about 6 minutes, and a report less than one minute. This result was expected since the downloading speed is highly variable because of local connection conditions; postprocessing requires quite some time given the big amount of data it needs to process and upload to the database, while reports developed so far are rather simple and, therefore, their execution is not particularly demanding. Looking at the total number of tasks, we can see that the amount of downloader and postprocessor tasks are comparable[2], while report tasks are about a fourth if compared to the others. This is because

[2]The number of postprocessors is slightly greater than the number of downloader tasks since postprocessor tasks fail from time to time. Being software that we adopted as a black box, we restarted the postprocessing after its failure.

Park	Ride	# report tasks	Computational time
Park 0	5	1062	11h33’40”
Park 3	0	786	7h55’54”
Park 4	15	113	1h42’53”
Park 0	16	58	5h30’31”
Park 0	10	43	5h24’03”
Park 4	23	35	56’48”
Park 0	21	10	16’47”
Park 6	19	8	16’42”
Park 1	11	4	01’36”
Park 0	28	2	13’46”
Park 16	1	2	02”

TABLE II: Computational times per ride of report tasks

it is possible not to produce any report. Therefore, we further inspect how many rides had associated report tasks.

Table II reports the number of tasks and computational times per ride of report tasks. Out of 39 rides, only eleven have report tasks. In addition, five of them have at most 10 report tasks, while only three have more than one hundred tasks, with the notable example of ride 5, which has more than one thousand report tasks. This result underlines the fact that, even if available, the automatization of reports was applied only in a few rare cases. We discussed this issue with the IT department. It turned out that two main factors caused this result: (i) the computational time of reports is rather limited, while getting them correctly executing on the software architecture takes quite some time, and (ii) they need to be manually adapted to the specific rides and heavily debugged, since each ride has its customizations, and therefore they were mostly running manually after the upload of the data to the database. We believe that point (ii) applies to any IIoT scenario since industrial devices are usually customized. On the other hand, point (i) is mostly because the reports produced so far involve only data about a single day. Currently, the IT team is developing weekly, monthly, and yearly reports, already experiencing the need for automatization since their computation requires too much time and resources. However, this is left as an open question and future work.

Answer to RQ1: The brief answer is yes, an automatized architecture is an enabling feature for innovative manufacturing. The long answer is that our results show that it is definitely needed for data collection and processing, while it is still debatable if this is necessary for report processing. In addition, the application of machine learning algorithms is out of scope regarding this work, but it might be an additional step after postprocessing requiring automatization.

B. *RQ2: Scalability and Elasticity*

We focus our discussion about RQ2 on distinguishing between scalability (that is, the ability of the architecture to sustain an increasing amount of computation) and elasticity (that is, the ability to dynamically adjust resources in response to sudden changing amounts of computation). Figure 5 reports the amount of computation sustained in each week of 2023 and 2024. We notice that, while the overall amount of computation is quite high with a peak of over 100 hours of computation in a single week, on the other hand, the amount of computation did not increase week after week. Therefore, at first glance, scalability is not needed. On the other hand, the system was under experimentation during this period, and therefore applied only on a relatively small and predefined set of rides. In the long run, once the system is in production and sold to customers, we expect the number of rides to increase to one hundred times the number of rides the architecture applied in this period. Such an increase will be slow, with customers adopting it one after the other.

Figure 4 reports the spread of computation over different days of the week and hours of the day. In particular, the plot on the left reports the sum of the computational times per day of the week, the one on the center the number of tasks per weekday, and the one on the right the sum of the computational times per hour of the day. About weekdays, we can notice that usually there is quite more computation at the beginning of the week (more than 400 hours on Monday), while over the weekend, there are the lowest peaks (around 250 hours, that is, about 40% less than Monday). The gap between the number of tasks is less relevant. We argue this is because, while rides are quite active on weekends, the IT department is active only during the week. Therefore, several tasks were planned over the week, not the weekend, accumulating more data to download and process.

The gap between different times of the day is evident when we look at the computational time per beginning hour. In particular, 7 a.m. is by far the most busy time of the day, with more than double the hours of all the other times. In addition, about 90% of the computation is condensed between 6 a.m. and 19 a.m. These results show that an elastic architecture (aka, it can dynamically adapt the resources consumed by the system) is crucial in this context.

Answer to RQ2: The current experiments show that elasticity is needed and very relevant at the current stage of architecture experimentation, while there is not yet evidence about the need for scalability. Instead, there is no doubt that elasticity is an essential feature required in this context.

VII. Conclusion

We transformed a legacy IIoT architecture into a containerized, automated, scalable, and elastic system for intelligent manufacturing. We have transformed the IIoT system from a passive and manual monitoring tool into an adaptive, resilient, and intelligent manufacturing framework by embedding these features. We dissected the architecture and discussed a Containerized Workflow Engine. Through our analysis, we identified several limitations in the initial Docker-based approach, including challenges related to scalability and inefficiencies in handling dynamic workloads. To address these issues, we introduced a distributed architecture that significantly enhances flexibility and scalability, allowing the system to adapt to fluctuating production demands while minimizing resource overhead seamlessly. To meet the evolving needs of modern industry, we incorporated essential properties such as elasticity

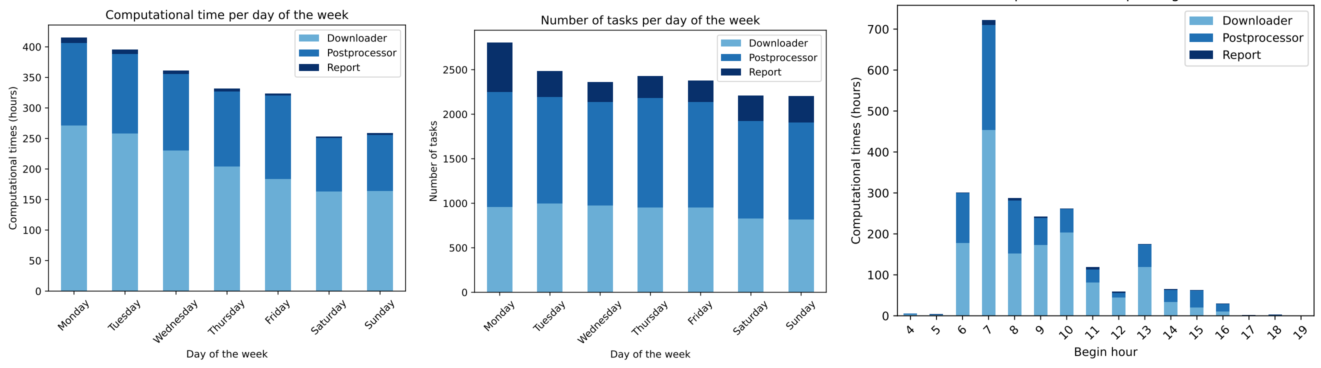

Fig. 4: Spread of computation over the week and the day

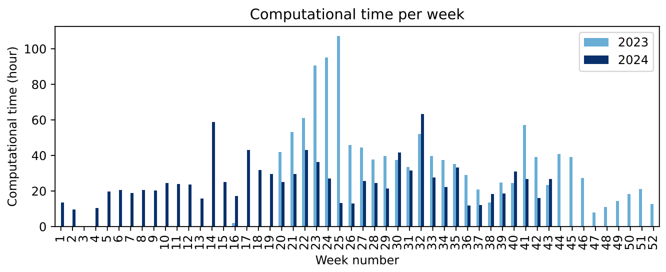

Fig. 5: Time of computation per week

and scalability alongside robust logging and accountability mechanisms that ensure full traceability and transparency.

The system has been fully developed and in operation for about one year and a half. The computational times extracted from the logs show unequivocally that automation and elasticity are two architectural characteristics that are inherently needed by IIoT architectures. Instead, the need for scalability has not yet been evident, but we expect this is due to the architecture's experimental adoption so far.

Future work The extreme flexibility of the system will allow the integration and coordination of additional modules in addition to data acquisition and analysis. The modules being defined will cover the monitoring of the integrity of the field installed components, communication performance, cybersecurity aspects and (last but not least) connections to digital machines models.

Acknowledgment Work partially supported by SERICS (PE00000014 – CUP H73C2200089001) under the NRRP MUR program funded by the EU – NGEU, by iNEST – Interconnected NordEst Innovation Ecosystem funded by PNRR (Mission 4.2, Investment 49 1.5) NextGeneration EU (ECS_00000043 – CUP H43C22000540006), and by the project RIR SUPREME.

References

[1] "Industrial internet of things: Unleashing the potential of connected products and services," World Economic Forum, Industry Agenda, Jan. 2015.

[2] G. Gokilakrishnan, V. M, A. Dhanamurugan, A. Bhasha, R. Subbiah, and A. H, "A Review of Applications, Enabling Technologies, Growth Challenges and Solutions for IoT/IIoT," in *2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS)*, vol. 1, Mar. 2023, pp. 2241–2250.

[3] G. Atharvan, S. Koolikkara Madom Krishnamoorthy, A. Dua, and S. Gupta, "A way forward towards a technology-driven development of industry 4.0 using big data analytics in 5G-enabled IIoT," *International Journal of Communication Systems*, vol. 35, no. 1, p. e5014, Jan. 2022.

[4] S. Abersfelder, E. Bogner, A. Heyder, and J. Franke, "Application and Validation of an Existing Industry 4.0 Guideline for the Development of Specific Recommendations for Implementation," *Advanced Materials Research*, vol. 1140, pp. 465–472, Aug. 2016.

[5] M. Khan, "Implementation of industry 4.0 smart manufacturing," Mar. 2019.

[6] P. Hu, "A System Architecture for Software-Defined Industrial Internet of Things," in *2015 IEEE International Conference on Ubiquitous Wireless Broadband (ICUWB)*. Montreal, QC, Canada: IEEE, Oct. 2015, pp. 1–5.

[7] J. Mora Abarca and L. Angulo Sánchez, "Determinación del nivel de conocimiento sobre Automatización que se requiere de los ingenieros industriales en Costa Rica," *Revista Fidélitas 2023 Volumen 4 Número 1*, vol. 4, no. 1, pp. 1–9, Jun. 2023.

[8] U. A. Alhaji and A. Rukaiya, "Investigation of the Level of Automation in Some Selected Manufacturing Industries in Kano, Nigeria," in *Proceedings of the International Conference on Industrial Engineering and Operations Management*. Nsukka, Nigeria: IEOM Society International, Apr. 2022, pp. 1–10.

[9] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu, "Edge Computing in Industrial Internet of Things: Architecture, Advances and Challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2462–2488, 2020.

[10] J. Jasperneite, T. Sauter, and M. Wollschlaeger, "Why We Need Automation Models: Handling Complexity in Industry 4.0 and the Internet of Things," *IEEE Industrial Electronics Magazine*, vol. 14, no. 1, pp. 29–40, Mar. 2020.

[11] T. Kurniawan, Y. Handayati, and G. Yudoko, "Exploring a Causal Loop Relationship between Lean Six Sigma and Insutry 4.0 Readiness: A Systematic Litearture Review," *Global Conference on Business and Social Sciences Proceeding*, vol. 15, no. 1, pp. 134–134, Sep. 2023.

[12] B. Ostadi, L. Barrani, and M. Aghdasi, "Developing a strategic roadmap towards integration in Industry 4.0: A dynamic capabilities theory perspective," vol. 208, p. 123679. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0040162524004773

[13] T. Goldschmidt and S. Hauck-Stattelmann, "Software Containers for Industrial Control," in *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 258–265. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7592806

[14] G. Di Modica and L. Foschini, “A Survey on the Use of Lightweight Virtualization in I4.0 Manufacturing Environments,” vol. 31, no. 2, p. 37. [Online]. Available: https://doi.org/10.1007/s10922-023-09725-4

[15] M. Basem, “Comparing plc, software containers and edge computing for future industrial use: a literature review,” 2022.

[16] E. S. Kumar, R. Ramamoorthy, S. Kesavan, T. Shobha, S. Patil, and B. Vighneshwari, “Comparative study and analysis of cloud container technology,” in *2024 11th International Conference on Computing for Sustainable Global Development (INDIACom)*, 2024, pp. 1681–1686.

[17] A. Alamoush and H. Eichelberger, “Open source container orchestration for industry 4.0 – requirements and systematic feature analysis,” *International Journal on Software Tools for Technology Transfer*, vol. 26, no. 5, pp. 527–550, Sep. 2024.

[18] D. K. Alqahtani and A. N. Toosi, “Container Orchestration in Heterogeneous Edge Computing Environments,” in *Resource Management in Distributed Systems*, A. Mukherjee, D. De, and R. Buyya, Eds. Springer Nature, pp. 151–168. [Online]. Available: https://doi.org/10.1007/978-981-97-2644-8_8

[19] Y. Wang, C. Lee, S. Ren, E. Kim, and S. Chung, “Enabling Role-Based Orchestration for Cloud Applications,” vol. 11, no. 14, p. 6656. [Online]. Available: https://www.mdpi.com/2076-3417/11/14/6656

[20] I. Korontanis, A. Makris, A. Kontogiannis, I. Varlamis, and K. Tserpes, “StreamK3s: A K3s-Based Data Stream Processing Platform for Simplifying Pipeline Creation, Deployment, and Scaling,” vol. 27, p. 101786. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711024001572

[21] J. Figueira and C. Coutinho, “Developing self-adaptive microservices,” vol. 232, pp. 264–273. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050924000267

[22] W. Bolton, *Programmable logic controllers*. Newnes, 2015.

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

[24] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, “Industrial internet of things: Challenges, opportunities, and directions,” *IEEE transactions on industrial informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[25] S. Sagiroglu and D. Sinanc, “Big data: A review,” in *2013 international conference on collaboration technologies and systems (CTS)*. IEEE, 2013, pp. 42–47.

[26] C. Li, Y. Chen, and Y. Shang, “A review of industrial big data for decision making in intelligent manufacturing,” *Engineering Science and Technology, an International Journal*, vol. 29, p. 101021, 2022.

[27] “Fast Innovation requires Fast IT,” 2014.