INTERNATIONAL SCIENTIFIC JOURNAL OF ENGINEERING AND MANAGEMENT ISSN: 2583-6129
VOLUME: 04 ISSUE: 03 |MAR – 2025 DOI: 10.55041/ISJEM02324
AN INTERNATIONAL SCHOLARLY || MULTIDISCIPLINARY || OPEN ACCESS || INDEXING IN ALL MAJOR DATABASE & METADATA

Integrating DevOps Principles into the Deployment Lifecycle of Distributed IoT Systems: Challenges and Best Practices

Naresh Kalimuthu
naresh.kalimuthu@gmail.com

Abstract*—Exploring how DevOps principles can enhance the deployment of distributed Internet of Things (IoT) systems offers a novel perspective, transcending traditional software delivery methodologies. This paper examines some of the primary challenges involved, such as the extensive diversity of physical devices, the unique security concerns related to physically dispersed devices, and the complexities of automated testing for systems integrating hardware and software. We identify three principal challenges: the suitability of CI/CD pipelines across various constrained and diverse hardware platforms, methods to automate end-to-end security (DevSecOps) on exposed surfaces susceptible to attacks, and the development of automated testing frameworks for hybrid systems. To address these challenges, we propose a comprehensive framework that incorporates Hardware Abstraction Layers (HALs) to manage device diversity, employs lightweight cryptography for security, and utilizes multi-tiered testing strategies through Hardware-in-the-Loop (HIL) simulation. Our case studies support the premise that overcoming these challenges can facilitate faster, more secure, and resilient IoT deployments.*

Keywords — DevOps, Internet of Things (IoT), CI/CD, Distributed Systems, Edge Computing, Firmware Deployment, IoT Security, Automated Testing, DevSecOps, Hardware-in-the-Loop (HIL).

I. INTRODUCTION

The Internet of Things, combined with the evolving 'DevOps' approach to software delivery and the rapid growth of IoT, represents two crucial shifts in current technology. DevOps aims to shorten the systems development cycle and enable continuous delivery of high-quality software. It fosters a culture that integrates various philosophies, practices, and tools related to system development. This culture emphasizes collaboration, automation, and quick feedback. By breaking down barriers between development and operations teams, organizations can improve how often and reliably they deploy software.

The growth of the Internet of Things (IoT) has brought about new, complex operational challenges in managing, updating, and protecting large, distributed systems and networks of interconnected devices, sensors, and actuators across industries like manufacturing, healthcare, agriculture, and transportation. Due to their size and complexity, deployments often become overly burdensome. Manual management approaches become increasingly vulnerable and ineffective, leading to operational inefficiencies, higher security risks, and higher project failure rates. Additionally, maintaining devices as marketable products becomes challenging, as tasks such as firmware updates, security patches, and system overhauls often become unfeasible. Therefore, integrating automation, continuous integration, and continuous delivery practices is crucial. These approaches make the complex IoT lifecycle more manageable. The operational difficulties and high failure rates of large IoT projects put significant pressure on businesses. Adopting DevOps practices is no longer just a technical choice but an economic necessity for survival and growth in a competitive IoT environment. Organizations that fail to automate their connected product lifecycle will struggle to scale, secure, and sustain these products, risking commercial failure.

A. Architectural Overview of Distributed IoT Systems: The Edge-to-Cloud Continuum

The IoT architecture is distinct and complex, and distributed devops can be a challenge. IoT differs from a traditional enterprise application, as they are not contained to a central datacenter. IoT systems are divided across a multi-layer architectural pattern IoT system is divided into three primary layers: the edge, the fog, and the cloud.

1) The Edge Layer:

This is the lowest layer of architecture, consisting of IoT devices such as physical sensors, smart appliances, and actuators that interact directly with the physical world. These devices are often limited in processing power, memory, and battery life. Edge computing is important for applications that are constrained by IPv4, require near real-time responses, and need low latency, such as industrial systems, distributed intelligent control modes, and autonomous systems.

2) The Fog / Gateway Layer:

Also known as the Gateway Layer, the fog layer in edge computing refers to a more geographically centralized region. It includes devices at one or more levels higher in the hierarchy, such as gateways or local edge fog servers. This layer primarily aggregates data from numerous edge devices. The main data sent is at Ethernet Level, with processing happening later before being forwarded to the cloud. This includes computers, lower fog level servers, cluster nodes, or fog nodes. Fog computing helps manage lower cloud control nodes and reduces network capacity requirements.

3) The Cloud Layer:

Cloud computing enables storing data in large repositories, analyzing it, and keeping it accessible for future use all in one place. It also allows for scaling tasks and performing complex computing jobs, such as training machine learning models or performing fleet-wide analytics, which fog or edge computing cannot handle.

The complexity of the architecture's structure and scope makes deployment very difficult. Even a simple feature update may require extensive coordination across all three layers, each with its own distinct hardware and software systems. The intricate interconnection of the architecture is the main reason why holistic deployment approaches often fall short, making automated deployment strategies essential.

B. *The Necessity of an Agile and Automated Framework for Distributed IoT Systems*

Managing the life cycles of distributed IoT systems is one of the most daunting challenges of the Internet of Things. Provisioning, transferring feature updates, deploying security patches, maintaining, and finally decommissioning millions of devices is nearly impossible due to the error-prone, inefficient, and sluggish process of relying on manual strategies. This realistic operational scenario further emphasizes the growing need for DevOps methodology.

1) *Continuous Integration (CI)*
This process involves the systematic and automated development and verification of software and its components, merging multiple separate instances of IoT software and firmware into a single consolidated version. This step ensures that automated builds of the software and firmware are accurately generated and validated. It is imperative to continually enhance and integrate the software and firmware versions.

2) *Continuous Deployment and Delivery (CD)*
This describes the process through which pre-production automated deployments are executed. It ensures that concise, over-the-air (OTA) software updates are delivered securely and efficiently to each device within the fleet. IoT software updates can be performed collectively, thereby minimizing human error and reducing operational costs.

3) *Infrastructure as code (IAC)*
This is the system responsible for managing and provisioning all aspects of the build infrastructure, testing rigs, and cloud services. It automates the process of defining infrastructure through machine-readable set files, thereby facilitating simpler and more straightforward configuration. Additionally, it ensures that no errors occur when handling the diverse range of devices within the system.

The adoption of these principles fundamentally shifts companies from a reactive stance to a proactive and adaptable posture, enhancing their capacity to automate processes more effectively. This transformation accelerates innovation while simultaneously improving the security and reliability of their IoT solutions. Such a transition necessitates that systems engineers expand their expertise to encompass not only hardware and firmware but also embedded systems, cloud-native solutions, automated processes, and security. This integrated expertise remains relatively rare within the industry.

C. *Key Challenges and Objectives*

While the necessity for DevOps in the Internet of Things (IoT) is evident, its practical implementation is complicated by a series of challenges that are not present in the conventional realm of software development. This paper aims to analyze these challenges systematically and to develop a set of evidence-based best practices to address them. Specifically, the focus is on how DevOps techniques such as continuous integration and automation can be effectively applied to managing resource-constrained and highly heterogeneous distributed IoT devices. Additionally, it examines the techniques required to achieve end-to-end security (DevSecOps) within the CI/CD pipeline, from code commit to firmware deployment on remote devices, which are often not only geographically dispersed but also physically insecure. Finally, the paper explores how the DevOps practice of automated testing, particularly for hybrid hardware and software systems—where interactions with the environment are critical for functionality—can be effectively implemented, despite the complexities involved in controlling such interactions. By systematically investigating these issues, this paper will establish a foundation for employing the DevOps approach throughout the deployment lifecycle of distributed IoT systems.

II. CORE CHALLENGES IN APPLYING DEVOPS TO IOT

Applying DevOps practices within the context of IoT technology is distinct from simply shifting practices in web development to embedded systems. IoT's more physical aspects and the constraints of architecture and resources challenge the foundational principles of DevOps. This subsection delineates the three primary challenges that are the focus of this research.

A. *Managing sets of modules and associated limited resources within an automated workflow*

The distinctive physical diversity of Internet of Things (IoT) devices constitutes the primary challenge. In a cloud environment, an average DevOps pipeline interacts with a collection of virtualized servers that are not physically connected. Conversely, an IoT pipeline must establish connections with an extensive and continuously expanding array of devices. This heterogeneity manifests in various forms, including processor architectures (such as ARM, MIPS, x86), RAM and flash storage capacities, operating systems (such as Linux, FreeRTOS, Zephyr), and a broad spectrum of communication protocols (including WiFi, Bluetooth, LoRaWAN).

The CI/CD pipeline encounters challenges stemming from diversity. The build process involves multiple cross-compilation toolchains tailored for various architectures. Validation must verify functionality across numerous hardware variants during testing. Deployment requires managing different firmware packages for distinct devices. Moreover, devices are afflicted by significant resource constraints, including limited processing power, memory, and battery life. These limitations impose constraints on size, efficiency, and software capabilities. Executing complex processes such as on-device testing or security protocols poses additional difficulties. The discrepancy between the software domain and the physical realm, as outlined in the concept of friction within the world, exemplifies the obstacle to successful adoption in the context of embedded hardware.

B. *Securing the Entire Lifecycle from Deployment to Development*

DevOps now encompasses both development and cloud infrastructure simultaneously, thereby minimizing the potential attack surface. However, with IoT, the attack surface extends far beyond the hypervisor and includes all physical components, stretching an order of magnitude greater. Every single element of the pipeline, stretching from a developer's workplace to the fielded device, has the capacity to be compromised.

Malicious code may be inserted into a firmware update through a vulnerability in a source code repository, a misconfiguration in a continuous integration (CI) server, or a compromised build tool. Such an update could potentially be propagated via an automated continuous deployment (CD) pipeline to a fleet of devices, leading to service disruptions, data breaches, or even physical harm in the context of critical infrastructure or medical devices. The over-the-air (OTA) update mechanism is frequently exploited as an attack vector; however, it remains one of the most neglected mechanisms. Furthermore, due to inadequate physical security, Internet of Things (IoT) devices are highly susceptible to tampering, which is often employed to extract cryptographic keys or alter device identities. Consequently, the DevSecOps approach to security becomes increasingly vital, emphasizing the integration of automated security practices at every stage of the pipeline, from code commitment to device provisioning and firmware updates deployment.

C. *Comprehensive Validation of Cyber-Physical Systems*

The deployment of hybrid hardware-software systems is typically intricate and fraught with challenges, particularly concerning the implementation of automated testing procedures. This complex functionality is occasionally termed "DevOps". It is generally characterized by immediate feedback derived from multiple tests conducted on the developed software. These tests are categorized into units, integration, and end-to-end assessments to optimize the virtual environment.

This specific case does not fit within the IoT context. While software logic can still be tested through unit tests, these do not verify the crucial interaction between software and hardware. Accessing physical hardware becomes the main bottleneck during integration and system-level testing. Such access issues are particularly problematic at these higher testing levels. Deploying physical hardware for the required tests is primarily time-consuming, difficult to automate, and nearly impossible to scale. When hardware isn't available, testing becomes scarce and costly. This hardware shortage tends to create queues and delays, disrupting the vital feedback loops in DevOps. Testing updates in a closed, active system without full hardware access increases risks—for example, missing issues or errors that only appear when the system is open, resulting in potential failures when the system is finally released.

III. RESEARCH AND RECOMMENDATIONS: STRATEGIES FOR MITIGATION OF DEVOPS IN IOT

There is a need for tailored strategies and focused approaches to address the complex problems arising from applying DevOps to IoT in automation, involving software and underlying embedded systems. This section provides a detailed explanation of the challenges and offers a comprehensive set of recommendations and mitigation strategies, grounded in evidence.

A. *Tackling Hardware and Resource Constraints*

The primary challenge faced by IoT in the context of DevOps pertains to managing an array of hardware types. Unlike server-side development, where applications are incorporated into a relatively standardized and virtualized environment, IoT solutions function across distributed, heterogeneous fleets. This diversity constitutes the most significant barrier to achieving universal integration within the Continuous Integration/Continuous Deployment (CI/CD) pipeline. Each type of hardware necessitates its own compiler, specific libraries, and customized build configurations. Moreover, these devices possess considerable limitations in terms of CPU capacity, power consumption, and memory resources. For instance, a microcontroller typically has minimal Random Access Memory (RAM) and operates on batteries. Consequently, resource-intensive or agent-based enterprise DevOps processes are impractical under such conditions. This necessitates that deployed systems be lightweight and energy-efficient. This challenge extends beyond mere technical considerations; it signifies a shift in values and organizational culture. Maintaining clean interfaces is imperative, and embedded teams adopting Standard Operating Procedures (SOP) are not exempt from these requirements—they must explicitly and carefully manage merging processes, boundary definitions, and logical dependencies.

Additionally, they must establish conditions conducive to reproducible environments. This constitutes the minimum criterion for successful integration within DevOps practices. Recent innovations in hardware "softwarization" have expanded technological capabilities by facilitating greater participation of software developers in the IoT domain.

To mitigate the complexity challenge, it is advisable to segregate the core application logic from hardware-specific details. The 'Hardware Abstraction Layer (HAL)" is a standardized software layer that acts as an Application Programming Interface (API) for peripheral hardware components such as sensors and control devices, thereby supporting various hardware peripherals. Most hardware-dependent applications are built upon different HAL schemes and remain portable across a wide range of Microcontroller Units (MCUs). Only the low-level HAL implementation requires adaptation for each new hardware target, greatly simplifying the CI pipeline. This modular approach not only makes development more efficient but also eases testing.

Principles of Infrastructure as Code (IaC), along with associated management strategies, are highly effective in addressing the complexities of multi-target builds. For embedded systems, this involves establishing distinct cross-compilation toolchains, SDKs, and dependencies tailored to each hardware platform. Within the Continuous Integration (CI) pipeline, tools such as Ansible, Chef, and basic shell scripting can automate the preparation of a clean build environment for each task, thereby ensuring consistency and repeatability. Containerization platforms like Docker are especially useful for encapsulating these environments. A Docker image can contain any toolchain and its dependencies within a portable, standalone container, thus providing a self-contained and isolated build environment irrespective of the host machine.

Continuous Deployment in IoT systems is implemented through Over-the-Air updates. Given that IoT devices often possess limited resources, these updates must be optimized for size and efficiency. Consequently, the utilization of delta updates is advisable, whereby only the binary differences between firmware versions are transmitted. Such delta updates can decrease the update payload size by over 90%, thereby expediting the update process, enhancing reliability in unstable network conditions, and improving overall power efficiency. These delta packages should be generated systematically, and staged rollouts ought to be managed by the Continuous Deployment (CD) pipeline, which facilitates monitoring of update success rates and enables automatic rollback in case of deployment failures, thereby ensuring the fleet's health and operational continuity.

B. Securing the IoT Deployment Pipeline (DevSecOps)

The interconnected nature of the Internet of Things (IoT expands the attack surface of the DevOps pipeline significantly beyond the walls of the data center. The implications of a security breach at any stage of the lifecycle—from code development to field deployment—can be devastating. The most critical aspects include compromised source code, insecure third-party components, pipeline poisoning, insecure Over-the-Air (OTA) transmissions, and physical device breaches. Successful breaches enable adversaries to gain control over entire fleets, leading to data theft, service disruption, and physical destruction. The Continuous Integration/Continuous Deployment (CI/CD) pipeline has undergone substantial transformation and is now regarded as the primary mechanism for security maintenance throughout the product lifecycle. The remaining segments of the product pipeline, including Continuous Delivery (CD), must also be trustworthy and secure; failure to ensure this elevates the product to a strategic business risk classified at the C-level. A compromised pipeline could result in widespread device bricking and extensive data breaches, incurring severe financial and reputational loss. This calls for investment in traditional web application DevOps toward enhancing pipeline security and resilience.

The primary emphasis of DevSecOps is on security and its incorporation at each phase of the software development lifecycle. Shift-left approaches integrate automated security checks within the CI/CD pipeline to deliver prompt feedback to developers. Key automated strategies include Static Application Security Testing (SAST) for examining application source code for vulnerabilities, Software Composition Analysis (SCA) for cross-examining software dependencies for known vulnerabilities (CVEs), and secret scanning to identify and flag sensitive data such as API keys, passwords, and tokens prior to commits, thereby preventing accidental exposure.

For a secure IoT system, a minimal assumption is that every device is assigned a unique and trusted identity, typically rooted in a cryptographic key stored in a secure hardware element and provisioned during the manufacturing process. The deployment system should authenticate this identity before pushing an OTA update to prevent unauthorized devices from accessing the network. The CD pipeline is required to merge with this identity system in order to employ a code-signing service that cryptographically signs the firmware artifact, as a demonstration of origin and integrity.

Indeed, the field of lightweight cryptography is of utmost importance due to the fact that standard cryptographic algorithms tend to consume a large volume of computational resources, especially on low-resource devices. The first option to consider is for the central distribution pipeline to employ a lightweight digital signature algorithm, such as those that derive from elliptic curve cryptography, to sign firmware packages for OTA updates. Once the device receives an update, it utilizes the relevant public key for signature verification to confirm that, prior to firmware installation, the signature has not been altered and is from a reliable source.

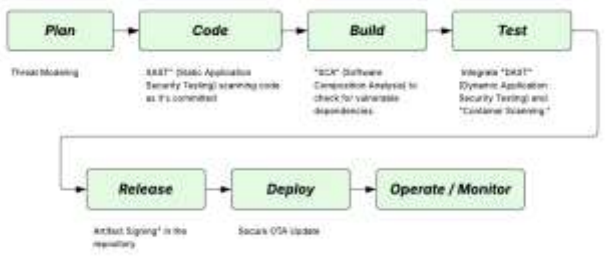
Plan
Code
Build
Test
Threat Modeling
SAST" (Static Application Security Testing) scanning code as it's committed
"SCA" (Software Composition Analysis) to check for vulnerable dependencies
Integrate "DAST" (Dynamic Application Security Testing) and "Container Scanning"
Release
Deploy
Operate / Monitor
Artifact Signing" in the repository
Secure OTA Update

Figure I. IOT-CENTRIC DECSECOPS PIPELINE (A "SHIFT LEFT" APPROACH)

C. Automating Testing for Complex IoT Systems

DevOps depends on rapid, automated feedback, primarily provided through a comprehensive suite of automated tests. In the context of Internet of Things (IoT) systems, the software testing process proves to be of limited utility as it largely neglects the hardware component. While unit tests can ascertain the logic of software modules, they are incapable of detecting bugs stemming from physical hardware interactions. Within a Continuous Integration (CI) pipeline, system-level tests that necessitate physical hardware are recognized as exceedingly difficult to automate. These tests tend to be slow, challenging to parallelize, and require expensive physical test laboratories. The absence of hardware remains one of the primary impediments to implementing Continuous Delivery in embedded systems. This hardware testing bottleneck impedes the rapid feedback cycle critical for Continuous Delivery, consequently delaying the overall development process.

Using virtualization technology is crucial to overcoming hardware limitations. Virtualization allows for quick and large-scale testing without physical systems. These tests include Software-in-the-Loop (SIL) emulation, where embedded software is compiled and run on a host machine, and system emulation, where tools like QEMU execute the actual cross-compiled firmware binary in a virtual environment. Although QEMU emulation is slower than SIL, it offers much better scalability and speed compared to testing on physical hardware.

While simulation and emulation are useful tools, they cannot fully replicate all the nuances of actual hardware. This underscores the significance of Hardware-in-the-Loop (HIL) testing, where the Device Under Test (DUT) is connected to a real-time simulator that mimics authentic sensors and environmental conditions. Automating HIL integration within the CI/CD pipeline involves creating a test harness that enables the CI server to handle firmware flashing, run specific test scenarios, evaluate results, and generate reports. Although setting up a HIL system can be costly, it offers increased confidence before deployment. Automated HIL testing covers real-world conditions, such as firmware integrity, hardware driver interactions, and power consumption. Incorporating Hardware-in-the-Loop (HIL) into CI/CD pipelines leads to "Test Infrastructure as Code." This emerging discipline requires engineers to possess a hybrid skill set, blending traditional test engineering, software automation, and DevOps expertise.

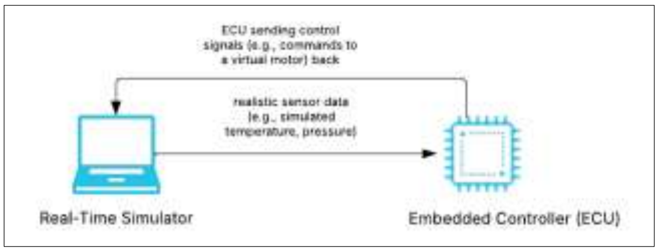
ECU sending control signals (e.g., commands to a virtual motor) back
realistic sensor data (e.g., simulated temperature, pressure)
Real-Time Simulator
Embedded Controller (ECU)

Figure II. HARDWARE-IN-THE-LOOP(HIL) TESTING SETUP

IV. EMPIRICAL EVIDENCE FROM REAL-WORLD APPLICATIONS

This section substantiates the proposed strategies through an examination of fundamental DevOps implementations and an analysis of specific IoT case studies. These cases underscore the observable benefits of adopting an automated, collaborative, and continuous lifecycle management methodology.

A. Lessons from Foundational DevOps Implementations

Blueprints for modern DevOps practices are based on the successes of web-scale giants like Amazon and Netflix. Amazon's decentralized, self-sufficient teams, empowered to make frequent independent changes, benefited from the microservices architecture. Their 'you build it, you run it' approach, along with significant cultural and structural shifts, became key to agility. Likewise, Netflix pioneered testing system resilience in production by intentionally introducing faults, a practice known as 'Chaos Engineering.” This required automated Continuous Delivery systems that help systems withstand failures by encouraging developers to create fault-tolerant systems from the start. The main lesson from these pioneers is that successful DevOps adoption hinges on fostering a culture of ownership, automation, and continuous improvement—essentials for success, even in IoT.

B. Analyzing IoT-specific implementations

1) Automotive and Fleet Management:

The adoption of DevOps for the Internet of Things has been vigorously pursued within the automotive sector. A leading global automotive manufacturer has modernized its legacy Retailer Report Portal by transitioning from a monolithic architecture to a scalable, cloud-native application hosted on AWS. This transition enables the automation of the entire Continuous Integration and Continuous Deployment (CI/CD) pipeline through GitHub Actions, Terraform, and Docker. The organization achieved a 56-fold increase in deployment frequency, and 82% of page load times were improved. Similarly, a provider of a fleet management solution designed a DevOps automation workflow to address challenges such as intermittent network connectivity and device vibrations in moving vehicles, utilizing Jenkins for continuous integration and Chef for configuration management. The implementation resulted in a 40% reduction in deployment efforts and a 90% increase in test coverage.

2) Automated Deployment for Fleet Management (eInfochips):

A manufacturer of in-vehicle camera systems experienced significantly prolonged release cycles, compounded by intermittent cellular data availability due to the dynamic environment of moving vehicles. Regrettably, the sporadic nature of cellular networks, coupled with the physical context of the moving vehicle, severely disrupted the release process. Utilizing Jenkins for continuous integration, Chef for configuration management, and automated testing frameworks, the organization automated the entire verification and health monitoring procedures to achieve comprehensive physical vehicle cycle automation. The resulting health monitoring systems, deployed with CI/CD practices, demonstrated notable improvements, including a 40 percent reduction in deployment effort and a 90 percent increase in coverage of automated tests. This example underscores the importance of implementing automated pipelines in remote IoT systems.

3) Applications in Agriculture and Healthcare:

The influence of DevOps on the Internet of Things (IoT) is increasingly evident within the healthcare sector. Surgical IoT devices play a vital role in real-time remote monitoring and intelligent patient care beyond traditional clinical devices. This category of automated surgical instruments connects various IoT devices with autonomous skin systems dedicated to Patient Safety IoT devices.

Figure III. CHALLENGES ENCOUNTERED, DEVOPS-CENTRIC SOLUTIONS IMPLEMENTED, AND IMPROVEMENTS REALIZED.

Organization / Industry	**Primary Challenge(s) Addressed**	**DevOps/IoT Solution Applied**	**Key Outcomes / Improvements**
Global Automotive Manufacturer	Outdated monolithic architecture; slow, manual deployments; poor scalability.	Modernized application with microservices on AWS; implemented full CI/CD automation with GitHub Actions, Terraform, and Docker.	56x increase in deployment frequency; 82% faster page load times; 65% reduction in database costs.
eInfochips Client (Fleet Mgmt)	Long release cycles; errors due to intermittent network connectivity and physical vibrations.	Automated CI/CD pipeline (Jenkins, Chef) for deployment, verification, and monitoring of in-vehicle camera firmware.	40% saving in deployment effort; 90% improvement in test coverage; quicker defect identification.
Coca-Cola Bottling Co. United	Complex, manual 11-step ordering and invoicing process that was unscalable and error-prone.	Implemented Microsoft Power Automate RPA integrated with Azure DevOps for a full CI/CD workflow.	Avoided hiring 10 FTEs; streamlined and automated the entire order-to-invoice process; enabled rapid scaling of a strategic product line.
SIG (Industrial Mfg)	Disparate data sources from production machines; lack of operational transparency.	Adopted PTC ThingWorx IoT platform to connect machines and visualize data on a real-time dashboard.	Increased visibility into production bottlenecks; identified and reduced unnecessary energy consumption; improved speed KPIs.

V. CONCLUSION

This paper systematically analyzes the adoption of DevOps practices within the deployment lifecycle of distributed IoT systems. It confirms that although adoption is essential for management and scalability, it is hindered by the hybrid nature of hardware and software components. The main challenges include hardware limitations, significant resource constraints, an extensive and physically accessible attack surface, and the paradox of testing automation for physical devices that are inherently hard to examine. This research provides a framework of suitable measures to address these challenges. The heterogeneity of hardware is managed through strategic abstraction and Infrastructure-as-Code (IaC) to create consistent engineering environments. DevSecOps promotes smoother adoption and moves security testing earlier via automated security scans, secure remote device provisioning, and lightweight encryption at the device layer to reduce security risks. Testing of integrated devices is separated through scalable simulation (SIL) supported by high-fidelity Hardware-in-the-Loop (HIL) techniques and other methods. The case studies verify that these practices effectively support DevOps adoption, improving agility, security, and reliability in IoT deployment.

REFERENCES

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in Proc. First Edition of the MCC Workshop on Mobile Cloud Computing, 2012, pp. 13–16. [Online]. Available: https://dl.acm.org/doi/10.1145/2342509.2342513

[2] J. Jung, B. Kim, J. Cho, and B. Lee, "A Secure Platform Model for Low-End IoT Devices with ARM Platform Security Architecture," IEEE Internet of Things Journal, vol. 9, no. 6, pp. 4773–4786, Mar. 2022. Available: https://web.engr.oregonstate.edu/~benl/Publications/Journals/IEEE_IoT_Journal_2022.pdf.

[3] Óscar López, "Devsecops methodology for NG-IoT ecosystem development lifecycle - assist-IoT perspective", J. Comput. Sci. Cybern., vol. 37, no. 3, p. 321–337, Sep. 2021.

[4] Bijwe, Awantika & Shankar, Poorna. (2022). Challenges of Adopting DevOps Culture on the Internet of Things Applications - A Solution Model. 638-645. 10.1109/ICTACS56270.2022.9988182.

[5] AWS Well-Architected Framework, "IoT Lens: Change Management," Amazon Web Services. [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/iot-lens/change-management.html.

[6] El Jaouhari, Saad. (2022). Toward a Secure Firmware OTA Updates for constrained IoT devices. 1-6. 10.1109/ISC255366.2022.9922087.

[7] Revolutionizing Banking: Jason Simon Champions Agile and DevOps for Today's Financial Landscape - Jason Simon. https://www.jason-simon.com/news/revolutionizing-banking-jason-simon-champions-agile-and-devops-for-todays-financial-landscape/