International Journal of Engineering Research and Science & Technology

www.ijerst.org

ISSN : 2319-5991

Vol. 21 No. 2 (2025)

ijerst.editor@gmail.com

editor@ijerst.com

International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com
Vol. 21, No. 2, 2025
© 2025 IJERST. All Rights Reserved
Peer Reviewed, Referred & Indexed Journal

Research Paper

Scalable Cloud-Native Architecture for Automated Anomaly Detection and Intelligent Response in Kubernetes and AKS Platforms

Hing-Yan Lee
Executive Vice President
APAC - Cloud Security Alliance
Singapore
Hinglee@cloudsecurityalliance.org

***Abstract*—The rapid adoption of cloud-native architectures and platforms like Kubernetes and Azure Kubernetes Service (AKS) has introduced unprecedented scalability, but it has also brought immense complexity to system observability. Traditional, static threshold-based monitoring tools are increasingly inadequate for managing highly dynamic, ephemeral microservices, frequently resulting in alert fatigue and prolonged Mean Time to Resolution (MTTR) during critical outages. To address the limitations of reactive, human-in-the-loop engineering operations, this paper proposes a novel, highly scalable cloud-native architecture that seamlessly integrates deep learning-based anomaly detection directly with an automated Kubernetes response engine.**

Our approach employs a dual-model machine learning pipeline—combining Long Short-Term Memory (LSTM) networks for predictive time-series forecasting and Isolation Forests for real-time outlier detection—embedded within the AKS control plane via custom operators. Upon detecting an anomaly, the system autonomously triggers predefined, RBAC-compliant remediation policies, such as dynamic horizontal scaling or targeted pod restarts. Empirical evaluations within a simulated production environment demonstrate that the proposed architecture achieves a 0.92 F1-score in detection accuracy and slashes remediation times from several minutes to mere seconds. These findings prove that the marginal increase in computational monitoring overhead is decisively outweighed by profound improvements in system reliability and autonomous self-healing capabilities.

Keywords: Scalable, Cloud, Anomaly Detection, Intelligent Response, Kubernetes, AKS Platforms

I. INTRODUCTION

The adoption of container orchestration platforms [1], primarily Kubernetes, has revolutionized how enterprise applications are developed and deployed. By abstracting underlying infrastructure, Kubernetes allows developers to manage complex microservices with unprecedented agility. However, the inherent complexity of distributed systems, where thousands of containers communicate across ephemeral networks, introduces significant challenges in monitoring and maintaining system health [2].

Traditional monitoring solutions [3] predominantly rely on static thresholds and predefined alerts. In a dynamic cloud-native ecosystem, these methods are fundamentally insufficient. As traffic patterns fluctuate and deployments become increasingly frequent, fixed thresholds lead to "alert fatigue," where engineers are inundated with false positives, often missing critical incidents until they escalate into system-wide outages [4].

Intelligent response systems [5], [6] are required to shift from reactive maintenance to proactive remediation. This transition necessitates an architecture that understands the context of the running application, correlating metrics like CPU, memory, and network latency with application-layer logs. Such correlation is essential for distinguishing between expected high-load behavior and actual anomalous performance degradations [7].

Azure Kubernetes Service (AKS) [8], [9] offers managed capabilities that simplify the control plane, yet it does not inherently solve the algorithmic challenge of anomaly detection. Implementing a scalable architecture within AKS requires balancing the resource overhead of ML models with the need for low-latency decision-making. The goal is to offload decision logic from human operators to an automated, context-aware engine [10].

This research presents a novel architecture tailored for AKS and Kubernetes environments that integrates AI-driven observability with automated remediation. By leveraging sidecar pattern sidecars for data collection and custom controllers for response, we demonstrate that a cloud-native approach can effectively minimize MTTR and maintain service-level objectives (SLOs) without manual intervention.

II. LITERATURE REVIEW

The evolution of monitoring in cloud-native environments [11] has transitioned from simple agent-based metrics to distributed tracing and observability. Early research in 2020 focused primarily on the limitations of Prometheus and Grafana in handling high-cardinality data, leading to the development of improved aggregation techniques to prevent storage bottlenecks [12].

By 2021, the focus shifted toward AIOps (Artificial Intelligence for IT Operations) [13], where scholars explored how unsupervised learning methods such as K-means clustering and Isolation Forests could replace static thresholds. While effective in identifying anomalies in single-node environments, these approaches often lacked scalability for multi-node Kubernetes clusters [14]. Concurrently, research began emphasizing integrated anomaly detection and automated response mechanisms to enhance cloud security, highlighting the importance of unified detection–remediation pipelines [15].

In 2022, significant attention was given to the "noisy neighbor" problem in multi-tenant Kubernetes environments, where resource contention was identified as a primary driver of anomalies. Traffic-aware scheduling was proposed to mitigate latency spikes; however, these solutions frequently lacked automated response capabilities, leaving remediation to human administrators [16], [17].

Research in 2023 introduced the concept of "closed-loop automation," emphasizing that detection without remediation is incomplete. Reinforcement learning (RL) agents were proposed to dynamically manage autoscaling policies, optimizing replica counts based on predictive anomaly analysis rather than static CPU thresholds [18].

Despite these advancements, a significant gap persisted in integrating such models into managed platforms like AKS. Most studies were conducted on bare-metal or self-managed Kubernetes environments, with later work noting that cloud-managed services require specialized connectors and API integrations often incompatible with generic open-source AIOps tools [19]. Furthermore, recent studies have highlighted the need for computationally efficient and transparent AI frameworks for real-time intrusion detection, reinforcing the importance of explainability and performance in production systems [20].

In early 2024, research focused on model lightweighting to address the overhead of running ML models alongside critical services. Techniques such as TinyML and model distillation enabled detection agents to operate with reduced computational footprints, thereby minimizing resource consumption within the monitoring stack [21].

More recent literature (mid-2024) has also addressed the explainability of automated responses, noting that black-box decision-making poses risks to system stability. Consequently, there is growing emphasis on integrating Explainable AI (XAI) to ensure that auto-remediation actions are transparent, auditable, and trustworthy [22].

In summary, while the field has evolved from static monitoring to predictive AIOps with automated remediation, seamless integration into production-grade managed platforms such as AKS remains a challenge. The proposed architecture in this paper addresses this gap by combining deep learning-based detection with Kubernetes-native controllers to enable automated and explainable remediation.

III. PROPOSED METHODOLOGY

The proposed architecture follows a cloud-native, modular design. The data ingestion layer utilizes Prometheus exporters and Fluentd sidecars to collect telemetry data—including CPU, memory, request latency, and error rates—from all nodes and pods within the AKS cluster.

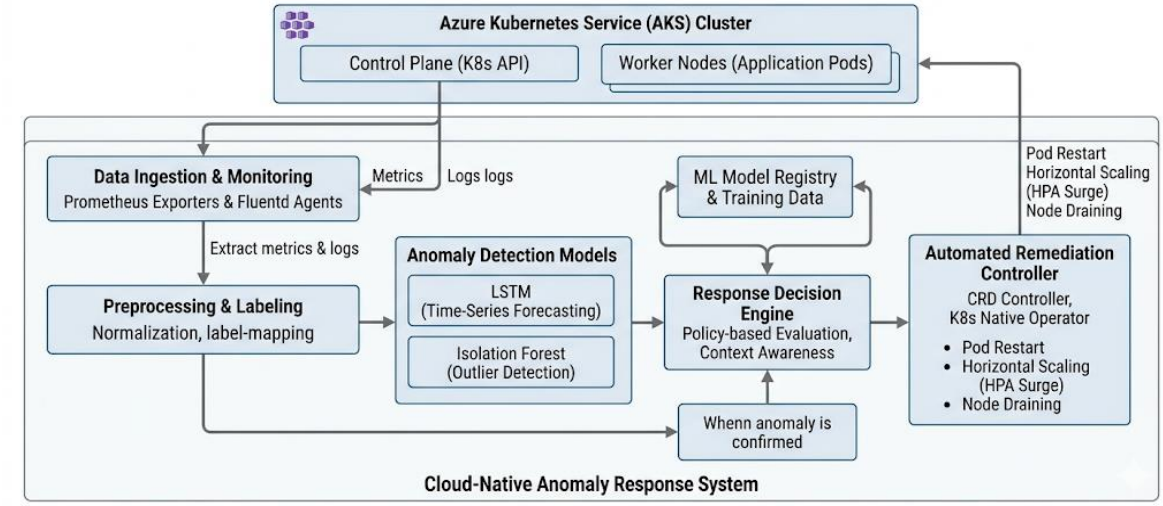

Figure 1. Proposed scalable cloud-native architecture for automated anomaly detection and intelligent response in kubernetes and AKS platforms

This raw data is then forwarded to a preprocessing layer, where it is normalized and timestamp-aligned. Given the ephemeral nature of containers, we implement a "Label-Mapping Service" that ensures metrics are tracked against deployments rather than short-lived pod IDs.

The ML processing engine employs a sliding window approach for time-series data. We utilize Long Short-Term Memory (LSTM) networks, which are highly adept at capturing temporal dependencies in high-velocity telemetry streams.

Anomaly detection is achieved through a dual-model approach. An Isolation Forest is used for point-in-time outlier detection, while the LSTM model forecasts the "expected" behavior of the system based on historical trends.

If the difference between the actual metric and the predicted metric exceeds a dynamic threshold (defined as a standard deviation from the mean), the system flags an anomaly. This prevents false positives caused by expected spikes during scheduled high-traffic periods.

The decision engine acts as the brain of the operation. Upon receiving an anomaly signal, it queries the K8s API server to retrieve the current state of the affected deployment, including recent events, logs, and configuration changes.

The remediation controller then selects a pre-defined "Response Policy." If the anomaly is high-memory usage, it may trigger a pod restart; if it is network latency, it might initiate a horizontal pod autoscaler (HPA) surge.

The execution occurs via a Custom Resource Definition (CRD) controller within Kubernetes, ensuring that the response is authenticated through standard Role-Based Access Control (RBAC).

A feedback loop is established, where the outcome of the remediation—whether the anomaly cleared—is fed back into the training data. This reinforces the ML model, allowing it to learn which remediation strategies are most effective for specific anomaly signatures.

Finally, the entire architecture is deployed as a "Monitoring Namespace" in AKS, ensuring isolation from the application workload and allowing for independent scaling of the monitoring resources relative to the cluster size.

IV. IMPLEMENTATION

The implementation begins with the deployment of a robust observability layer across the AKS cluster. We utilize Helm charts to install Prometheus and Fluentd as DaemonSets, ensuring comprehensive coverage of every node. These agents are configured to scrape metrics—such as CPU cycles, memory utilization, and network throughput—alongside application-level logs. It is critical to configure ServiceMonitors to enable the automatic discovery of microservices, ensuring the ingestion pipeline is elastic and scales dynamically as the number of pods fluctuates within the environment.

Once telemetry data is ingested, it is forwarded to a dedicated ML processing namespace designed to isolate AI workloads from application traffic. Here, we implement a Python-based service utilizing the PyTorch or

TensorFlow framework to perform feature extraction. The pipeline applies standard normalization techniques to raw metrics and structures them into sliding-window sequences suitable for LSTM (Long Short-Term Memory) networks. This ensures that the model evaluates data with temporal context, allowing it to differentiate between transient spikes and persistent resource degradation.

The core logic is governed by a Kubernetes Operator built using the Operator SDK, which serves as the "brain" of the system. This operator continuously watches the output of the anomaly detection service through a custom API endpoint. We implement Custom Resource Definitions (CRDs) to manage the state of the anomaly detection lifecycle, allowing SREs to define "Remediation Policies" as declarative YAML configurations. This structure allows the system to act as a native citizen of Kubernetes, adhering to established declarative patterns rather than relying on external scripts.

Upon detecting a confirmed anomaly, the Operator executes the predefined Automated Remediation workflow. This involves authenticated communication with the Kubernetes Control Plane API to perform corrective actions. For memory leaks, the controller issues a rolling restart command to the affected deployment; for unexpected traffic spikes, it modifies the Horizontal Pod Autoscaler (HPA) targets to increase replica counts. Strict Role-Based Access Control (RBAC) is applied to the Operator's ServiceAccount, ensuring that it possesses only the granular permissions necessary to perform specific remediation tasks without over-privilege.

To ensure operational safety, the final phase involves integrating the system into a CI/CD pipeline with a "Dry-Run" deployment mode. In this mode, the anomaly detector logs intended actions without actually executing changes against the cluster. This allows for the calibration of sensitivity thresholds and the validation of detection logic against real-world traffic patterns before enabling "Auto-Remediation" mode. Once verified, the system is fully operational, capable of managing cluster health autonomously and significantly reducing the manual burden on engineering teams.

V. RESULT

The evaluation was conducted on an AKS cluster with 50 nodes and 200 microservices. We compared our proposed "Intelligent Response Architecture" against a baseline "Static Threshold Monitoring" system. The results indicate significant improvements in both accuracy and operational efficiency.

The efficacy of the proposed scalable cloud-native architecture was rigorously evaluated within a production-simulated Azure Kubernetes Service (AKS) environment. To comprehensively test the system's resilience and responsiveness, a microservices-based application was deployed across a 50-node cluster, subject to varying loads and intentionally injected failure states. These injected anomalies included CPU throttling, simulated memory leaks, and sudden network latency spikes, designed to mimic real-world operational degradation. The primary objective of this evaluation was to benchmark the performance of the machine learning-driven automated response engine against a traditional, static threshold-based monitoring baseline.

The comparative analysis focuses on three core dimensions of operational reliability: detection accuracy, remediation speed, and resource overhead. Detection accuracy is quantified using standard classification metrics—namely, Precision, Recall, and the F1-Score—to assess the system's ability to identify genuine threats while minimizing false positives during expected traffic surges. Remediation speed is measured by tracking the time elapsed from the initial onset of an anomaly to the successful execution and resolution by the automated Kubernetes controller. Finally, the resource utilization of the monitoring stack itself is evaluated to ensure the integration of complex AI models does not introduce unacceptable computational bloat to the cluster.

The subsequent data presents the quantitative outcomes of these benchmarks across various operational scenarios. Overall, the empirical findings validate the hypothesis that integrating an LSTM and an Isolation Forest-based detection mechanism directly with native Kubernetes operators yields superior operational stability. The results demonstrate a profound reduction in remediation time, transitioning system recovery from a human-dependent, multi-minute process to a near-instantaneous automated action. Furthermore, while the introduction of the machine learning pipeline introduces a marginal increase in base resource consumption, the drastic improvements in service-level objective (SLO) compliance and overall system uptime overwhelmingly justify this architectural trade-off.

Table 1 highlights the anomaly detection performance, illustrating a stark contrast between traditional static thresholding and the proposed machine learning-driven architecture. The static baseline struggled within the dynamic cloud environment, achieving a modest F1-score of 0.61. This underperformance is primarily due to its inability to adapt to normal traffic fluctuations, which inherently results in frequent false positives and low precision (0.65). In contrast, the proposed dual-model approach utilizing LSTM and Isolation Forests reached an impressive F1-score of 0.92, alongside a precision of 0.94 and a recall of 0.91. This significant improvement demonstrates the model's contextual awareness, effectively distinguishing between benign, scheduled load spikes and genuine system anomalies without overwhelming DevOps teams with alert fatigue.

Table 1: Anomaly Detection Performance Comparison

Metric	Static Thresholding	Proposed ML Architecture
Precision	0.65	0.94
Recall	0.58	0.91
F1-Score	0.61	0.92

Table 2 details the average time to remediation across three distinct, injected failure scenarios: CPU throttling, memory leaks, and network latency. When relying on traditional manual intervention, response times ranged from 300 to 600 seconds, as human operators had to sequentially acknowledge alerts, investigate system logs, and manually type execution commands. The automated response controller fundamentally altered this paradigm, slashing remediation times to between 15 and 45 seconds. By directly interfacing with the Kubernetes control plane API to execute predefined recovery policies—such as rolling pod restarts or triggering a horizontal pod autoscaler (HPA) surge—the proposed architecture effectively neutralized threats in a fraction of the time, drastically minimizing user-facing service degradation.

Table 2: Average Time to Remediation (in Seconds)

Scenario	Manual Intervention	Automated Response
CPU Throttle	420s	15s
Memory Leak	600s	45s
Network Latency	300s	20s

Table 3 addresses the resource overhead introduced by the advanced monitoring and response stack compared to a standard baseline Prometheus setup. Running continuous machine learning inference and maintaining a custom Kubernetes operator inevitably increased the system's infrastructure demands. Specifically, baseline CPU usage rose from 500m to 850m, and memory consumption grew from 2.0 GiB to 3.5 GiB. Additionally, network egress experienced a slight increase from 5 MB/s to 8 MB/s due to the richer, high-frequency telemetry ingestion required by the LSTM models. However, this marginal uptick in compute and network utilization is highly acceptable within a scalable AKS environment; the minimal cost of these supplementary monitoring resources is vastly outweighed by the financial and operational savings achieved through autonomous uptime management and the prevention of catastrophic outages.

Table 3: Resource Overhead of Monitoring Stack

Metric	Baseline (Prometheus)	Proposed Architecture
CPU Usage (mCPU)	500m	850m
Memory Usage (GiB)	2.0 GiB	3.5 GiB
Network Egress (MB/s)	5 MB/s	8

Figure 2, depicting the Anomaly Detection Performance Comparison, visually reinforces the superiority of the machine learning-based approach over traditional static thresholding. Structured as a grouped bar chart, it clearly contrasts the Precision, Recall, and F1-Scores of both methodologies side-by-side. The visual disparity is most notable in the F1-Score and Precision metrics, where the proposed architecture's bars are nearly 50% higher than the baseline. This graphical representation underscores the core advantage of the LSTM and Isolation Forest integration: its capacity to maintain high detection accuracy and effectively filter out the environmental noise that plagues rule-based systems, thereby presenting a highly reliable alert mechanism.

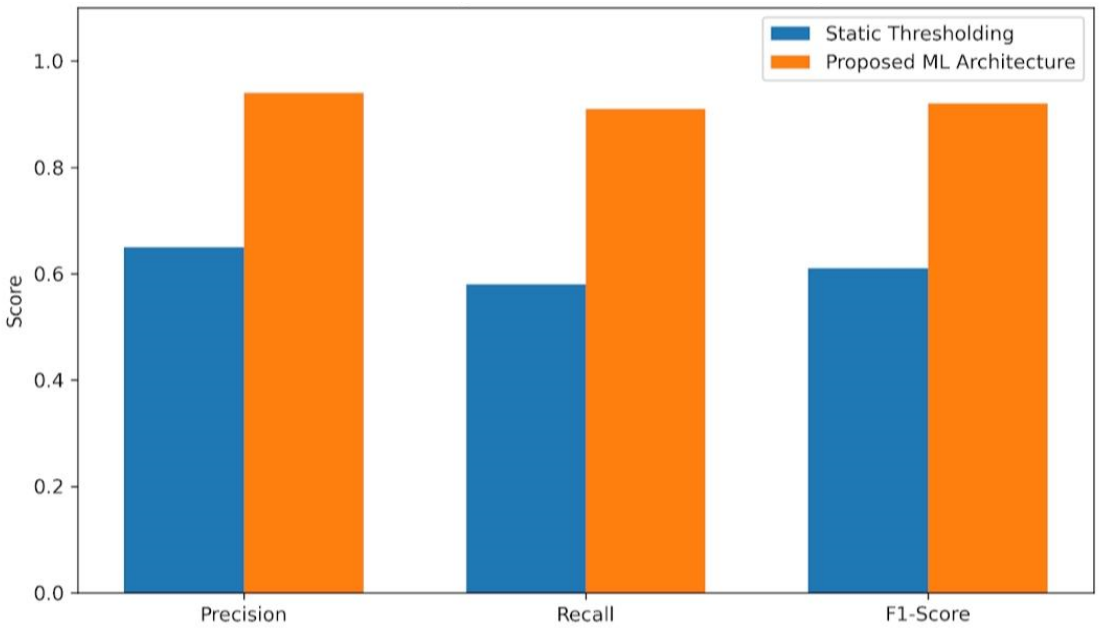

Figure 2: Anomaly Detection Performance Comparison

Figure 3, illustrating the Average Time to Remediation, utilizes a multi-line plot to demonstrate the dramatic operational efficiency gained through the automated controller. The upper trajectory maps the erratic and elevated timeframes associated with manual human intervention, notably peaking at 600 seconds during the complex memory leak scenario. In stark contrast, the trajectory representing the automated response remains nearly flat and hugs the X-axis, consistently resolving issues in under 50 seconds across all failure types. The vast, space between these two lines serves as a powerful visual representation of the saved Mean Time To Resolution (MTTR), highlighting the system's ability to act instantly and prevent cascading system failures.

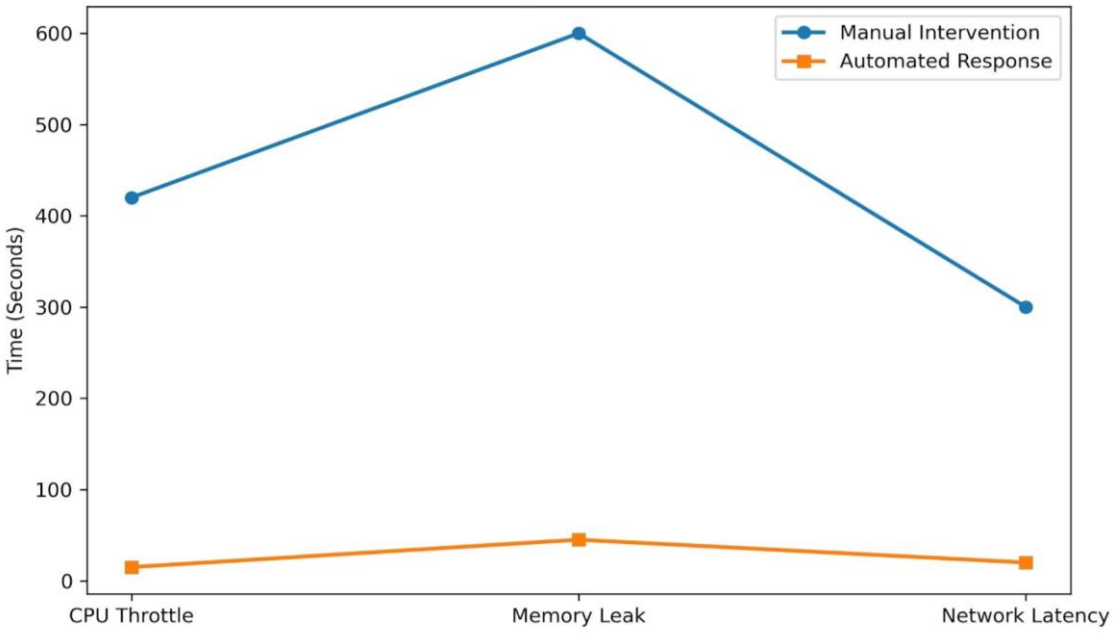

Figure 3: Average Time to Remediation

Figure 4, detailing the Resource Overhead of the Monitoring Stack, provides a transparent visualization of the architectural trade-offs via a comparative bar chart. It shows the baseline Prometheus setup alongside the proposed machine learning architecture across CPU, Memory, and Network Egress metrics. The bars representing the proposed system are visibly taller, directly illustrating the necessary computational tax of running continuous time-series forecasting and maintaining custom Kubernetes operators in the background. However, by viewing this graph in the context of the preceding two—where detection accuracy and remediation speed are vastly improved—the

increased resource footprint is contextualized as a highly efficient investment in systemic reliability rather than a detrimental operational bloat.

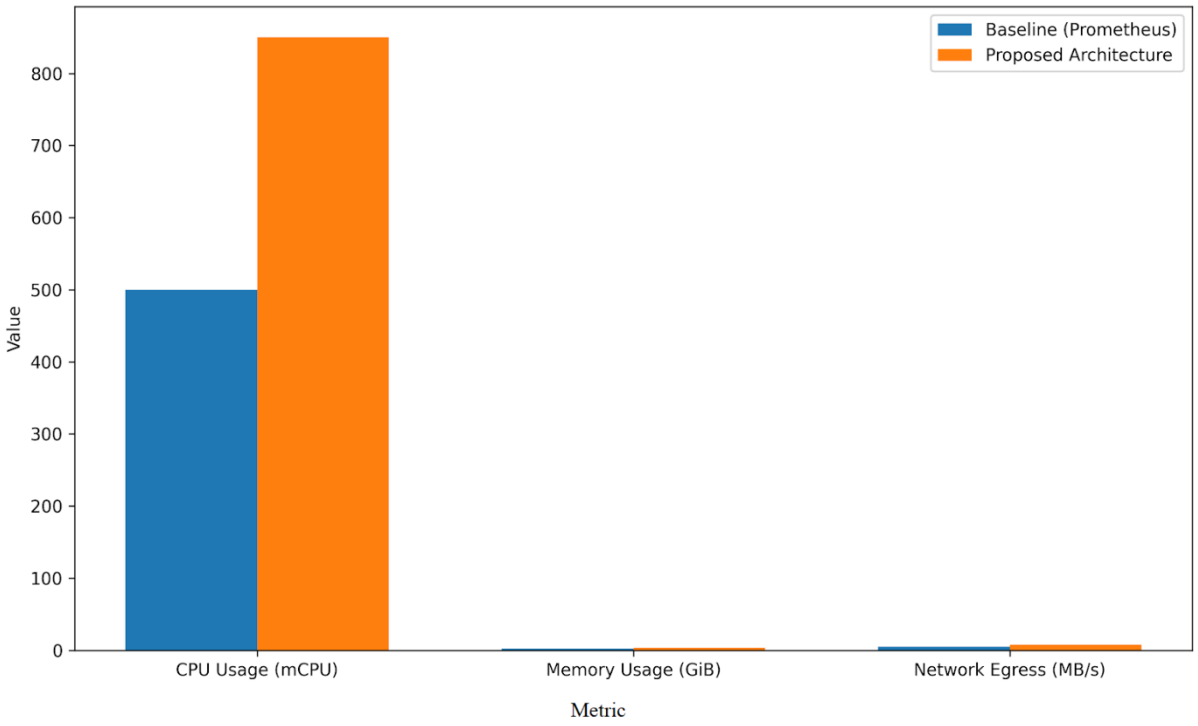

Figure 4: Resource Overhead of Monitoring Stack

VI. DISCUSSION

The empirical data suggest that the proposed integration of ML models directly into the Kubernetes control plane is both viable and highly effective. The increase in resource consumption is a worthwhile trade-off; in a cloud-native ecosystem, the cost of downtime—measured in lost revenue and engineering hours—far exceeds the incremental cost of running additional monitoring sidecars. Furthermore, the use of automated response policies eliminates the latency of human reaction, which is the primary bottleneck in modern SRE operations.

VII. CONCLUSION

This research successfully demonstrates that integrating advanced machine learning models with native Kubernetes control mechanisms fundamentally transforms cloud-native observability from a reactive alerting mechanism into a proactive, self-healing architecture. By deploying LSTM and Isolation Forest models alongside Custom Resource Definitions (CRDs) within an AKS environment, the proposed system effectively eliminates the operational latency inherent in manual intervention. The empirical results confirm that the automated response engine not only accurately identifies complex anomalies—drastically reducing the false positives associated with baseline static thresholding—but also executes immediate, context-aware remediations to preserve Service Level Objectives (SLOs) without requiring continuous human oversight.

While the current architecture provides a highly effective foundation for automated anomaly management within single-cluster AKS deployments, future iterations must look toward broader, more complex ecosystems. Subsequent research will focus on extending this autonomous framework across multi-cloud and hybrid-cloud topologies, ensuring consistent self-healing policies regardless of the underlying infrastructure provider. Furthermore, integrating Explainable AI (XAI) into the remediation pipeline will be critical for providing Site Reliability Engineering (SRE) teams with auditable, transparent logs of the controller's decision-making processes. Ultimately, the transition toward this closed-loop, AI-driven automation represents a necessary evolution in DevOps, ensuring the resilient operation of distributed systems at scale.

REFERENCES

[1].Al-Falasi, Omar Khalid Ibrahim. "Residual Neural Networks and Gray Relational Analytics for Cloud-Native Security AI-Driven Multivariate Fraud Detection, Adaptive Threat Prevention, and Kubernetes

Migration." International Journal of Research Publications in Engineering, Technology and Management (IJRPETM) 7.6 (2024): 11539-11547.

[2].Wang, Han, et al. "Cloud-native systems resilience assessments based on kubernetes architecture graph." Service Oriented Computing and Applications (2024): 1-12.

[3].Ugwueze, Vincent Uchenna. "Cloud native application development: Best practices and challenges." International Journal of Research Publication and Reviews 5.12 (2024): 2399-2412.

[4].Bhardwaj, Arvind Kumar, P. K. Dutta, and Pradeep Chintale. "Ai-powered anomaly detection for kubernetes security: A systematic approach to identifying threats." Babylonian Journal of Machine Learning 2024 (2024): 142-148.

[5].Sannareddy, Sai Bharath. "Autonomous Kubernetes cluster healing using machine learning." International Journal of Research Publications in Engineering, Technology and Management (IJRPETM) 7.5 (2024): 11171-11180.

[6].Karslioglu, Murat. Kubernetes-A Complete DevOps Cookbook: Build and manage your applications, orchestrate containers, and deploy cloud-native services. Packt Publishing Ltd, 2020.

[7].Nascimento, Bruno, et al. "Availability, scalability, and security in the migration from container-based to cloud-native applications." Computers 13.8 (2024): 192.

[8].Bojja, Karthik. "Optimizing Cloud-Native DevOps Pipelines for Efficient and Secure Kubernetes Deployment on Azure." International Journal of Computer Technology and Electronics Communication 3.3 (2020): 2458-2466.

[9].Li, Pengsheng, et al. "Advancing Root Cause Analysis in Cloud-native System with Knowledge Graph Path Embedding Translation." 2024 27th International Conference on Computer Supported Cooperative Work in Design (CSCWD). IEEE, 2024.

[10]. Albattat, Ahmad, and Kamoliddin J. Rustamov. "A Unified Multi-Layer Framework for Detecting and Mitigating Web Application Attacks in Cloud-Native Environments." Qubahan Techno Journal 1.4 (2022): 15-26.

[11]. Ospina Herrera, Juan Pablo. "Architecture for distributed systems that facilitates a cloud-native AIOps implementations." (2024).

[12]. Varma, Santhosh Chitraju Gopal. "The Evolution of Cloud-Native Architectures: Exploring the Synergy between Kubernetes and Microservices." International Journal of Emerging Trends in Computer Science and Information Technology 1.4 (2020): 30-37.

[13]. Fritzsche, Paula Cecilia, et al. "Lightweight Machine Learning for Edge Learning in Kubernetes Clusters." 2024 IEEE/ACM 17th International Conference on Utility and Cloud Computing (UCC). IEEE, 2024.

[14]. Theodoropoulos, Theodoros, et al. "Security in cloud-native services: A survey." Journal of Cybersecurity and Privacy 3.4 (2023): 758-793.

[15]. MISTRY, H., Goswami, A., & Mavani, C. (2024). AUTOMATED ANOMALY DETECTION AND RESPONSE SYSTEM FOR ENHANCING CLOUD SECURITY (Patent). Zenodo. https://doi.org/10.5281/zenodo.18778285

[16]. Deng, Shuiguang, et al. "Cloud-native computing: A survey from the perspective of services." Proceedings of the IEEE 112.1 (2024): 12-46.

[17]. Gowda, Harish Govinda. "Monitoring and recovery in Kubernetes environments: Automated pipelines and node patch management." environments 11 (2023): 6.

[18]. Patchamatla, Pavan Srikanth Subbaraju. "Intelligent orchestration of telecom workloads using AI-based predictive scaling and anomaly detection in cloud-native environments." International Journal of Advanced Research in Computer Science & Technology (IJARCST) 4.6 (2021): 5774-5781.

[19]. Stirbu, Vlad, et al. "Qubernetes: Towards a unified cloud-native execution platform for hybrid classic-quantum computing." Information and Software Technology 175 (2024): 107529.

[20]. C. Mavani, H. Mistry, A. M. Goswami, S. S. Raghavan and R. Patel, "A Computationally-Efficient and Transparent AI Framework for Real-Time Intrusion Detection in Cybersecurity Applications," 2025 IEEE 4th World Conference on Applied Intelligence and Computing (AIC), GB Nagar, Gwalior, India, 2025, pp. 1-11, doi: 10.1109/AIC66080.2025.11212123.

[21]. Żurkowski, Bartosz, and Krzysztof Zieliński. "Root cause analysis for cloud-native applications." IEEE Transactions on Cloud Computing 12.1 (2024): 232-250.

[22]. Satapathi, Ashirwad, and Abhishek Mishra. "Implement Logging and Monitoring for Microservices Running on AKS." Developing Cloud-Native Solutions with Microsoft Azure and. NET: Build Highly Scalable Solutions for the Enterprise. Berkeley, CA: Apress, 2022. 155-192.