Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

A Structured Framework For Autonomous Failure Prediction And Self-healing In Microservices

B. Harish Goud, N. Sudhakar Yadav, P. Kiranmaie, Kandhuri Sai Rasagna, Battiprolu Sai Jeevan
Department of Information Technology
Chaitanya Bharathi Institute of Technology (CBIT), Hyderabad, India

***Abstract*—Microservice-based distributed networks are popular due to their ability to expand and always be available. Yet, they may also introduce such issues as the slow performance, downtimes, and breach of Service Level Agreements (SLAs). The existing procedures do not offer a definite solution to address these issues and respond primarily to the fact that the problem has already happened, which delays the recovery process and reduces the efficiency of the system. This paper is a well-articulated failure management framework that can be used to anticipate failures, investigate their root causes, and support automatic recovery with limited human intervention. The system uses a closed-loop pipeline with steps for observing, predicting, reasoning, taking action, verifying, and constantly updating. The observation layer gathers telemetric metrics in real time, every moment, and the predictive layer finds potential challenges before they actually happen. An agentic AI-based reasoning layer makes decisions based on the situation to choose the right recovery actions.The action layer does these actions in a shadow-mode environment. The verification stage checks how the framework works after it has been run and makes changes to the framework as needed. The proposed strategy is a complete unified application to help build smart, self-adaptive microservice architectures and make systems more robust.**

***Index Terms*—Microservices, Anomaly Detection, Self-Healing Systems, Failure Prediction, AIOps, Cloud-Native Systems, Fault Tolerance, Distributed Systems, Self-Adaptive Systems**

I. Introduction

Modern distributed microservices systems are popular because they can be easily scaled and are always available. Their modular design lets you install services separately and make good use of resources. However, the increasing number of interconnected components adds significant complexity to regulating overall system behavior.

In these situations, common difficulties include performance problems, service interruptions, and breaking Service Level Agreements (SLAs). It's difficult to find those flaws in microservices because they are spread out across several services, which might cause failures to spread. Most of the ways we monitor systems today are reactive, which means they only find problems after they happen. This makes recovery take longer and makes the system less reliable.

A complete framework that integrates intelligent decision-making, automatic recovery, and failure prediction is absent from current systems. This study suggests an organised architecture for autonomous failure prediction and self-healing in microservices systems in order to overcome these constraints. Proactive failure management, enhanced system resilience, and reduced downtime are made possible by the suggested approach's closed-loop procedure, which consists of steps for observation, prediction, reasoning, action, and verification.
The primary contributions of this paper are as follows:

- A structured closed-loop architecture for autonomous failure prediction and self-healing in microservice-based environments.
- Unified system architecture that incorporates autonomous recovery, context-aware reasoning, and failure prediction.
- Introduction of an agentic AI-based reasoning layer that makes it possible to make context-driven, intelligent decisions on the best course of action for recovery.
- The incorporation of a controlled execution mechanism via shadow-mode operation, which is employed during initial deployment or low-confidence settings to validate prescribed actions prior to execution, assuring safe and dependable autonomous operation.

II. Literature Survey

The extensive use of cloud-native microservices has made managing contemporary distributed systems much more difficult. As the number of services and their interactions grow, maintaining system stability and ensuring reliable performance becomes more challenging. Researchers have looked into a number of strategies to deal with these issues, mostly concentrating on automated recovery techniques, monitoring, and predictive analysis. However, rather than being a part of a coherent system, these tactics are often developed as standalone solutions.

Initially, monitoring-based methods that depend on gathering logs, metrics, and traces from dispersed components were used to address system reliability. Prometheus and the ELK Stack are well-liked technologies for tracking system behaviour and spotting unusual patterns. [1].

To improve early detection, a number of studies have created machine learning-based methods that analyse system data to identify warning signs of potential malfunctions. Important performance metrics (latency, CPU usage, and error rates) are used by Random Forest and Support Vector Machines to identify troublesome situations [2]. Additionally, deep learning models have been used to better understand temporal patterns in system activity, especially those designed for sequential

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

data. Methods based on hybrid architectures and LSTM have demonstrated improved capacities in predicting irregularities in systems [4] [5]. Despite these advancements, most of these solutions focus just on prediction and do not expand their capabilities to automated recovery.

In addition to prediction models, self-healing methods have been developed to automatically respond to system faults. To guarantee system uptime, container orchestration solutions - Kubernetes in particular offer integrated capabilities like auto-scaling, health checks, and service restarts [3]. Additionally, recovery judgements have been automatically improved with the application of reinforcement learning techniques. These methods reduce the need for physical intervention, but because they are triggered after a failure has already occurred and do not employ predictive insights to prevent disruptions, they are usually reactive in nature.

More recent AIOps research has introduced agent-based methods that aim to support intelligent and context-aware decision-making within distributed systems. Without the need for human intervention, these systems evaluate system conditions and choose suitable corrective actions [6]. Nevertheless, a lot of these technologies are still in their infancy and do not work with execution environments like Kubernetes, real-time monitoring pipelines, or validation techniques like shadow-mode testing.

Overall, most solutions primarily address individual components, despite the fact that recent research has greatly improved several aspects of system stability. There is currently no comprehensive system that combines automatic recovery, prediction, monitoring, and decision-making into a single continuous workflow. This disparity highlights the need for a more comprehensive approach that can support proactive and independent system management in microservices-based configurations.

TABLE I
COMPARISON OF EXISTING APPROACHES

Ref	Method	Contribution	Limitation
[1]	ML-driven observation	Anomaly identification from logs and runtime data	Post-fault detection (no early prediction)
[2]	Classical ML (RF, SVM)	Uses QoS metrics to detect SLA violations	No automated recovery support
[3]	ML + RL (K8s)	Automated recovery and adaptive scaling	Only operates following failure detection
[4]	Deep Learning	Gets Patterns to Predict Failure	Unrelated to Execution Systems
[5]	LSTM-Transformer	Improves Temporal Forecasting Precision	Unsuitable for Cloud Operations in Real Time
[6]	Context-aware decision making	Agent-based AI	Absence of a single process or validation

III. PROPOSED FRAMEWORK

This paper highlights the transition in microservices architectures from reactive monitoring to proactive failure prediction and management [7, 8]. The recommended method combines predictive analysis with automatic recovery approaches to resolve system problems and Service Level Agreement (SLA) violations before they occur. It uses a structured closed-loop pipeline that continuously monitors telemetry data in real time, anticipates possible failures, chooses the best recovery plan, and confirms system behaviour following execution. Moreover, the framework is safe in the validation of activities that require such validation by offering limited execution through shadow-mode execution. The approach aims at enhancing the stability of microservices in dispersed and dynamic scenarios by incorporating various processes into a unified architecture.

A. Observation Layer

The observation layer receives the continuous telemetry feeds of microservices environment that contains important operational data such as resource usage, traffic volume, and service to service request patterns. To maintain the present status of the system, data is transferred with little latency instead of relying on delayed reporting. This continuously moving data can be taken as a solid basis to predict failure accurately as it makes it possible to recognize aberrant activity at an earlier stage and process it quickly at a later stage.

B. Prediction Layer

Rather than simply reacting to immediate alerts, the prediction component evaluates the patterns based on the observation layer in order to anticipate the possible issues before they arise and take preventive measures. It can identify small changes that can be indicative of failures by analyzing not only recent observations but also the activity of the systems previously trained (1). The system is able to act upon the early warning signs before the problems escalate to significant disruptions. It can therefore be possible to have a more reliable system management method since the entire system can remain in constant operation with fewer failures.

$$A(t) = |X(t) - \hat{X}(t)| \tag{1}$$

where $A(t)$ is the anomaly score, $\hat{X}(t)$ is the projected value, and $X(t)$ is the observed system metric.

C. Reasoning Layer

The reasoning layer serves as the framework's decision-making component, selecting appropriate remedial or preventative measures in response to expected failures or violations. It evaluates the output of the prediction layer and looks at the system environment to identify the root causes of abnormalities. This entails assessing factors including traffic patterns, resource usage trends, and recent system changes that may have influenced the reported behaviour. Unlike traditional rule-based or model-driven systems that could struggle to handle dynamic or unexpected situations, the suggested reasoning layer makes conclusions using a context-aware approach. It assesses a range of potential causes and chooses appropriate corrective measures based on the system's current state and previously observed system behaviour.

The outcomes of this layer include particular recovery actions and confidence scores, which show how reliable each

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

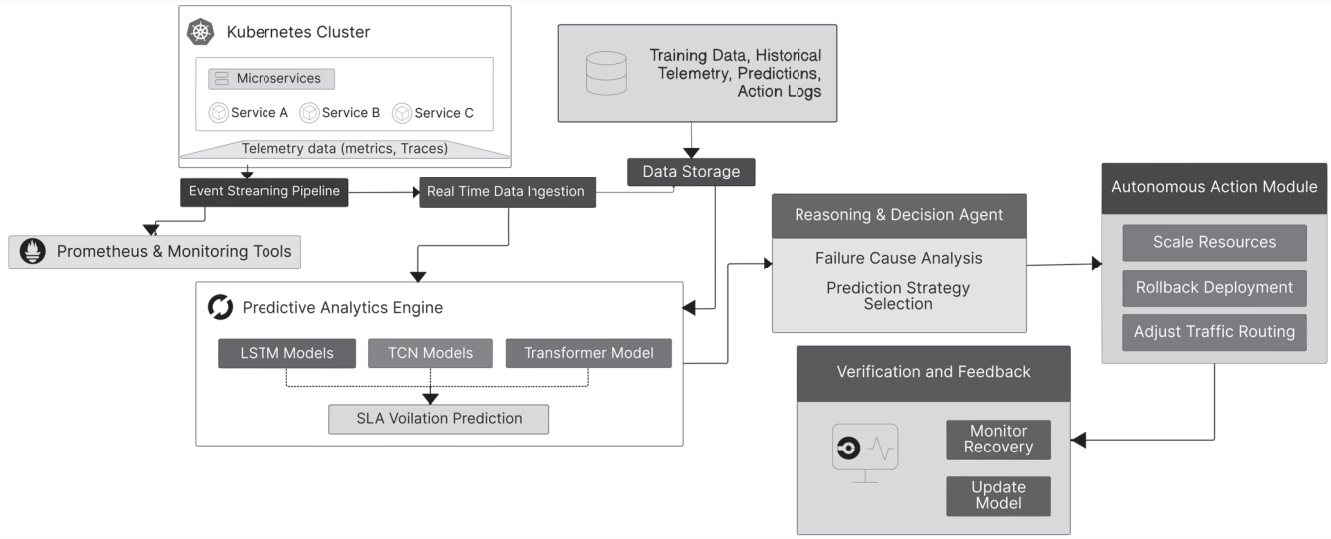

Fig. 1. Proposed Framework Architecture

choice is and influence the execution strategy in later stages, as shown in (2). Based on the corresponding confidence scores, the action layer implements decisions made by the reasoning layer.

$$C(a) = P(a \mid S) \tag{2}$$

where $C(a)$ is the confidence of action a, and $P(a \mid S)$ is the probability of action given system state S.

D. Action Layer

The action layer is required to implement the conclusions made by the reasoning layer based on the corresponding confidence scores.

As a result, the execution approach is chosen: autonomous execution of high-confidence tasks enables prompt response to expected failures. Conversely, lower-confidence operations are routed through a shadow-mode execution process, where they are recorded and, if desired, subject to human review prior to full deployment.

Therefore, the execution strategy is selected: autonomous execution of high-confidence tasks allows a quick reaction to anticipated failures. In contrast, operations that will have lower confidence are sent via a shadow-mode execution process, which they are logged and, optionally, can be reviewed by humans before full deployment.

E. Verification Layer

The verification layer determines the degree of success with which the decisions have brought the system back to its optimal state [9]. It gets positive feedback and is consideed to be the correct action to take when it is satisfactory. If it is not up to par, it is forwarded for further evaluation and negative system feedback. The experience gained on that component gets returned into the learning system in situations where human intervention was necessary ensuring it will be in a position to deal with a similar situation in future.

IV. Discussion

Embracing the aspects of observation, prediction, decision-making, and automatic response in a loop, the proposed architecture provides a comprehensive solution to the failur.e management in microservices-based environments [12]. The proposed architecture allows the different components to interact in a coordinated manner compared to the traditional systems that handle these phases independently, and therefore, the system will respond in a quicker and smarter manner.

One of the principal advantages of this method is the ability to study system behaviour without major issues surfacing. By examining both the already existing tendencies and the current activity of the system, the framework will be able to detect the signs of potential instability. This reduces the chances of service outages and improves the overall system reliability by allowing corrective actions to be done in advance.

It is complemented by a reasoning layer that enhances the framework and allows making decisions on a case-by-case basis [6]. The system takes the initiative to evaluate the character of the problem and select the most appropriate solution instead of relying on pre-determined actions or rules. It can more successfully manage a greater range of failure scenarios due to its versatility.

Another important element is the interface with orchestration platforms that enables the automatic execution of corrective measures: the transfer of workloads, the restart of the service, or the adaptation of the resources [7]. A feedback phase also ensures that the consequences of every activity are observed and used to enhance the response of the next action so that there is gradual change in the behaviour of the system with time.

Despite these benefits, there are several practical questions that should be considered. The predictive component reliability is affected by the relevance of the input data and the ability

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

of the models to adapt to changing conditions within the system. In complex environments, when many subsystems are combined, it could increase the overhead of the operations and would require close coordination to ensure efficiency. A reliable performance at scale and fast response times remains a major design challenge.

Altogether, the proposed paradigm demonstrates that the management of the system changes the reactionary approach to fault management to the more flexible and proactive one. It provides a foundation of the development of smart, self-adaptive microservices systems, which are capable of maintaining stability in changing environments, integrating a number of functional levels within a process.

V. Conclusion

This paper suggested a structured system integrating monitoring, prediction, decision-making, and automation transformation in one process to address the problems in microservices-based systems. The proposed architecture incorporates these capabilities instead of considering them as separate parts in order to make the systems activity more timely and coordinated.

To minimize unexpected service failures, the method constantly analyzes the activity of the system and seeks the first signs of instability. The reasoning layer adds to the flexibility of the system to various failure situations by enabling the system to make decisions based on the current situation and not predetermined rules.

Despite its conceptual nature, the framework provides a realistic path towards the development of self-managing systems with little involvement of human being.

References

[1] IRJMETS, “Self-Healing Microservices and AI-Based Anomaly Detection System,” 2025.
[2] IJERT, “Machine Learning for Real-Time SLA Violation Detection,” 2025.
[3] WJAETS, “AI-Enhanced Self-Healing Kubernetes Systems,” 2025.
[4] arXiv, “AI-Enhanced Fault Tolerance in Microservices,” 2024.
[5] PMC, “LSTM-Transformer Models for Time-Series Prediction,” 2024.
[6] arXiv, “Agentic AI for AIOps Systems,” 2025.
[7] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, “Borg, Omega, and Kubernetes,” *Communications of the ACM*, vol. 59, no. 5, pp. 50–57, 2016.
[8] J. Lewis and M. Fowler, “Microservices: A Definition of This New Architectural Term,” 2014.
[9] G. Chen, H. Jin, D. Zou, and H. Chen, “Automated Failure Diagnosis in Microservices Systems,” *IEEE Transactions on Cloud Computing*, 2022.
[10] M. Du, F. Li, G. Zheng, and V. Srikumar, “DeepLog: Anomaly Detection and Diagnosis from System Logs,” in *Proc. ACM CCS*, 2017.
[11] A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices Architecture Enables DevOps,” *IEEE Software*, vol. 33, no. 3, pp. 42–52, 2016.
[12] P. Chen *et al.*, “Towards Intelligent AIOps Systems: A Survey,” *IEEE Access*, 2020.
[13] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[14] A. Vaswani et al., “Attention Is All You Need,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
[15] B. Sigelman et al., “Dapper, a Large-Scale Distributed Systems Tracing Infrastructure,” *Google Technical Report*, 2010.
[16] B. Hindman et al., “Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center,” in *NSDI*, 2011.