

---

# ***Predictive Reliability Assessment Model for Containerized Enterprise Software Systems***

## **Abstract**

Containerized enterprise software systems require early reliability assessment because failures can develop through resource pressure, repeated restarts, slow dependencies, and unstable orchestration. This study presents a predictive reliability model that combines container restart behaviour, processor and memory pressure, service-response degradation, dependency health, and orchestration instability. Runtime metrics were collected across fixed monitoring windows and converted into temporal prediction features. The model was evaluated under six increasing failure-stress levels and compared with threshold-based monitoring and history-only assessment. Results showed that the proposed approach followed the observed system condition more closely and identified reliability decline earlier than the comparison methods. It also provided indicator-level warnings that explained the main causes of reliability loss. The framework can support proactive maintenance, faster operational decisions, and reduced service interruption by helping enterprise teams detect developing container failures before they become severe and affect dependent services across distributed production environments during demanding and rapidly changing workloads.

**Keywords:** Container reliability, Predictive assessment, Enterprise software, Failure prediction, Orchestration monitoring

## 1. Introduction

Containerized enterprise systems package application services and their required dependencies into independent runtime units. This design allows organizations to deploy, update, and scale services more quickly across cloud infrastructure. However, container failures can occur because of resource exhaustion, software errors, unstable dependencies, incorrect scheduling, or orchestration problems. Industrial microservice systems depend strongly on resource monitoring, load balancing, log management, and exception tracking to maintain reliable operation [1]. A predictive assessment model can use these operational signals to identify reliability decline before a major service interruption occurs.

Container orchestration platforms automatically start, stop, replace, and relocate containers according to application demand. These actions improve flexibility, but they also create rapidly changing system states that are difficult to evaluate using occasional health checks. Containerized microservice performance can change over time as workload, service population, and communication demand increase [2]. Reliability assessment should therefore examine trends across several monitoring intervals rather than treating each measurement as an isolated event. This wider view can reveal gradual degradation that may not immediately cross a fixed warning threshold.

Failures in containerized systems may also spread across connected services. A slow database container can increase the response time of several application services, while an unstable message broker can interrupt complete business workflows. Spatial and temporal relations among containers can be used to identify cascading fault paths and estimate how failures may spread through a cloud environment [3]. A predictive model should therefore consider dependency health together with the condition of individual containers. This approach can provide a more realistic estimate of enterprise system reliability.

Reliable operation also depends on how well the orchestration platform responds to changing workload and infrastructure conditions. Autoscaling may create too few containers, respond too late, or repeatedly increase and decrease the service population. Failure-prediction methods for autoscaling distributed applications can analyse runtime behaviour before performance and availability requirements are violated [4]. This study proposes a predictive reliability assessment model that combines restart behaviour, resource pressure, service-response degradation, dependency failure, and orchestration instability to estimate the future reliability of containerized enterprise software systems.

## 2. Methodology

The proposed methodology included monitoring-data collection, container-state processing, reliability-feature extraction, failure-risk prediction, reliability-score generation, and model evaluation. Runtime

information was collected from container engines, orchestration platforms, service monitors, distributed traces, application logs, and infrastructure metrics. The monitored variables included restart count, processor use, memory pressure, network delay, response time, failed requests, dependency status, replica availability, and scheduling events. Microservice anomaly-detection models can combine distributed traces with profiling metrics to represent both service relationships and runtime behaviour [5]. The proposed model used this combined evidence to describe the current operational state of every container.

The collected records were arranged into fixed monitoring windows and matched using timestamps, container identifiers, service names, and orchestration events. Missing values were estimated only when the interruption was short, while incomplete monitoring windows were excluded from model training. Resource values were normalized according to the capacity assigned to each container. Response-time prediction has been used to support proactive scaling decisions for microservice applications running in Kubernetes environments [6]. The same predictive principle was applied here to identify whether current operational trends were likely to reduce reliability during the next monitoring interval.

Five groups of prediction features were created from the processed monitoring data. Restart features included restart frequency, time since the last restart, and repeated failure patterns. Resource-pressure features represented processor saturation, memory limits, network delay, and storage activity. Service-response features measured latency growth, throughput reduction, timeout frequency, and failed-request ratio. Dependency and orchestration features represented unhealthy connected services, replica imbalance, failed placement, rescheduling, and scaling instability.

Table 1 presents the five reliability indicators used by the proposed model. Each indicator was linked with observable monitoring evidence and converted into one or more prediction features. The model assigned a larger failure risk when several related indicators changed together, such as increasing memory pressure followed by slow responses and repeated container restarts. Proactive container autoscaling can use attention-based recurrent learning to predict future resource use from earlier runtime observations [7]. This temporal approach supported the detection of developing reliability problems rather than only confirmed failures.

**Table 1. Container Reliability Indicators, Monitoring Evidence, Prediction Inputs, and Reliability Assessment Outputs**

| <b>Reliability indicator</b> | <b>Collected runtime evidence</b>                           | <b>Derived prediction feature</b>            | <b>Assessed failure risk</b>                 | <b>Generated reliability output</b>           |
|------------------------------|-------------------------------------------------------------|----------------------------------------------|----------------------------------------------|-----------------------------------------------|
| Container restart behaviour  | Restart count, exit code, uptime, and time between restarts | Restart frequency and repeated-failure trend | Risk of crash loops and service interruption | Container stability score and restart warning |

|                              |                                                                         |                                                         |                                                          |                                                            |
|------------------------------|-------------------------------------------------------------------------|---------------------------------------------------------|----------------------------------------------------------|------------------------------------------------------------|
| Resource pressure            | CPU use, memory use, storage activity, and network load                 | Resource saturation and pressure-growth rate            | Risk of throttling, exhaustion, or forced termination    | Resource reliability score and overload alert              |
| Service-response degradation | Response time, throughput, timeout rate, and failed requests            | Latency slope and service-quality decline               | Risk of performance failure and unavailable transactions | Service reliability score and degradation warning          |
| Dependency failure           | Health state, trace errors, connection failures, and downstream delay   | Dependency-failure probability and propagation strength | Risk of cascading failure across services                | Dependency reliability score and affected-service list     |
| Orchestration instability    | Failed scheduling, replica imbalance, rescheduling, and scaling changes | Placement instability and scaling-oscillation rate      | Risk of reduced capacity and unstable recovery           | Orchestration reliability score and recovery-status output |

The prediction engine processed the feature sequence using a supervised learning model trained on normal operation, gradual degradation, resource overload, dependency interruption, and orchestration-failure records. It estimated the probability of service failure within the next monitoring window and converted this probability into a reliability score. Machine learning-based infrastructure management can predict the changing resource needs of containerized applications and adjust capacity to reduce response-time violations [8]. The proposed model used similar runtime prediction but focused on reliability assessment rather than infrastructure cost alone.

The final system reliability score combined the five indicator scores using weights based on their operational impact. Confirmed dependency failures and repeated container restarts received greater weights because they could directly interrupt several enterprise functions. Resource and response trends received increasing weights when they continued across consecutive windows. Chaos-engineering experiments can measure the dependability of cloud-native applications by introducing delays and observing system behaviour under different user loads [9]. Controlled failure injection was therefore used to produce realistic evaluation scenarios and verify whether the predicted score followed the actual system condition.

The model was evaluated under six increasing container-failure stress levels. The scenarios included processor overload, memory pressure, repeated restarts, slow dependencies, replica loss, failed scheduling, and unstable autoscaling. The proposed predictive model was compared with threshold-based monitoring and history-only reliability assessment. Performance was measured using prediction accuracy, precision, recall, F1-score, reliability-score error, early-warning time, and failure-detection delay. The same containerized application, workload sequence, and injected failure conditions were used for every method to maintain fair evaluation conditions.

### 3. Results and Discussion

The proposed predictive model was evaluated across six increasing container failure-stress levels. Reliability scores were calculated for the proposed model, threshold-based monitoring, and history-only assessment under the same workload and failure conditions. At lower stress levels, all three methods reported high reliability because container restarts, resource pressure, dependency errors, and orchestration instability remained limited. As the stress level increased, the proposed model showed a gradual reliability decline that closely followed the actual condition of the containerized system.

Figure 1 shows the predicted reliability scores produced by the three assessment methods. The proposed model identified early degradation because it combined restart trends, resource pressure, response-time growth, dependency health, and orchestration behaviour. Threshold-based monitoring reacted later because it changed the reliability score only after fixed limits were crossed. History-only assessment also responded slowly because it depended mainly on earlier failure patterns and did not fully represent current runtime changes.

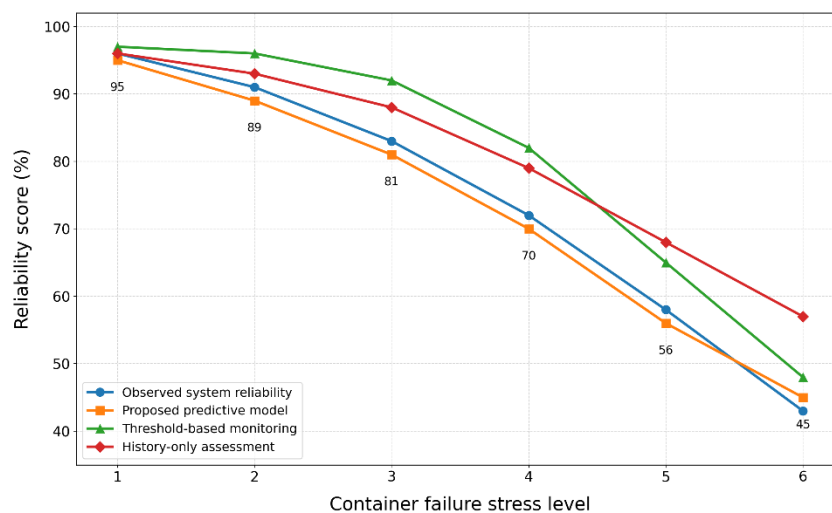


**Figure 1. Predicted Reliability Score Across Increasing Container Failure Stress Levels**

The largest difference appeared at the medium and high stress levels. Repeated container restarts, memory pressure, slow dependencies, replica loss, and unstable scaling reduced the observed system reliability. The proposed model reflected these changes within the next monitoring window and produced a smoother decline than the comparison methods. This behaviour shows that combining several runtime indicators can provide a more complete reliability estimate than checking separate alerts.

At the highest stress level, the proposed model remained closer to the observed containerized system condition. It provided earlier warning of possible service interruption and identified the indicators that contributed most to the reliability loss. Threshold-based monitoring underestimated the risk until several failures became severe, while history-only assessment could not respond quickly to new

orchestration problems. These results show that predictive reliability assessment can improve early warning, failure diagnosis, and operational planning for enterprise container platforms.

#### 4. Conclusion

This study presented a predictive reliability assessment model for containerized enterprise software systems. The model combined container restart behaviour, resource pressure, service-response degradation, dependency failure, and orchestration instability within one temporal assessment process. It used live monitoring data and earlier operational patterns to estimate future failure probability and generate a reliability score for every monitoring interval. The framework also produced indicator-level warnings that helped explain why reliability was increasing or decreasing.

The results showed that the proposed model detected reliability decline earlier and followed the observed system condition more accurately than threshold-based monitoring and history-only assessment. It responded effectively to overload, repeated restarts, dependency interruption, replica loss, failed scheduling, and unstable autoscaling. The proposed approach can therefore support proactive maintenance, faster operational decisions, reduced service interruption, and more reliable management of large containerized enterprise applications.

#### References

1. Waseem, M., Liang, P., Shahin, M., Di Salle, A., & Márquez, G. (2021). Design, monitoring, and testing of microservices systems: The practitioners' perspective. *Journal of Systems and Software*, 182, 111061.
2. Camilli, M., & Russo, B. (2022). Modeling Performance of Microservices Systems with Growth Theory: Modeling Performance of Microservices Systems with Growth Theory. *Empirical Software Engineering*, 27(2), 39.
3. Chen, N., Zhong, Q., Liu, Y., Liu, W., Bai, L., & Hu, L. (2023). Container cascade fault detection based on spatial-temporal correlation in cloud environment. *Journal of Cloud Computing*, 12(1), 59.
4. Denaro, G., El Moussa, N., Heydarov, R., Lomio, F., Pezzè, M., & Qiu, K. (2024). Predicting failures of autoscaling distributed applications. *Proceedings of the ACM on Software Engineering*, 1(FSE), 1960-1981.
5. Panahandeh, M., Hamou-Lhadj, A., Hamdaqa, M., & Miller, J. (2024). ServiceAnomaly: An anomaly detection approach in microservices using distributed traces and profiling metrics. *Journal of Systems and Software*, 209, 111917.

6. Pramesti, A. A., & Kistijantoro, A. I. (2022, September). Autoscaling based on response time prediction for microservice application in Kubernetes. In *2022 9th International Conference on Advanced Informatics: Concepts, Theory and Applications (ICAICTA)* (pp. 1-6). IEEE.
7. Dogani, J., Khunjush, F., & Seydali, M. (2022). K-agrued: A container autoscaling technique for cloud-based web applications in kubernetes using attention-based gru encoder-decoder. *Journal of Grid Computing*, 20(4), 40.
8. Shafi, N., Abdullah, M., Iqbal, W., & Bukhari, F. (2025). DIMA: machine learning based dynamic infrastructure management for containerized applications. *Computing*, 107(3), 88.
9. Al-Said Ahmad, A., Al-Qora'n, L. F., & Zayed, A. (2024). Exploring the impact of chaos engineering with various user loads on cloud native applications: an exploratory empirical study. *Computing*, 106(7), 2389-2425.