International Journal of Scientific Research and Engineering Development-– Volume 9 Issue 2, Mar-Apr 2026
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Intelligent On-Demand System Health Monitoring Using Golang and Prometheus Node Exporter

Arunkumar A*, Dr S Selvakani**, Mrs K Vasumathi***
*PG Research Scholar, Government Arts and Science College, Arakkonam
Email: ak738@gmail.com
**Assistant Professor and Head PG Department of Computer Science, Government Arts and Science College, Arakkonam
Email: sselvakani@hotmail.com

***Assistant Professor PG Department of Computer Science, Government Arts and Science College, Arakkonam
Email: kulirmail@gmail.com

----------**********----------

Abstract:

Modern systems require efficient monitoring to ensure reliability and performance. This paper presents an intelligent on-demand system health monitoring framework using Golang and Prometheus Node Exporter. The system collects CPU, memory, disk, and network metrics in real time, helping administrators monitor system performance efficiently.

Keywords **— System Monitoring, Golang, Prometheus, Node Exporter, System Health Monitoring, DevOps Monitoring.**

----------**********----------

1 INTRODUCTION

System monitoring is important for maintaining the performance and stability of modern computing environments. This paper proposes an intelligent on-demand system health monitoring approach using Golang and Prometheus Node Exporter to collect and analyse system metrics such as CPU, memory, disk, and network usage efficiently.

2 LITERATURE SURVEY

Recent studies highlight the importance of reliable system monitoring for maintaining performance in modern computing environments. Tools such as Prometheus and Node Exporter are widely used to collect system metrics including CPU, memory, disk, and network usage. Research also shows that Golang provides efficient and lightweight backend services for monitoring applications. These technologies support scalable and real-time monitoring solutions.

3 METHODOLOGY

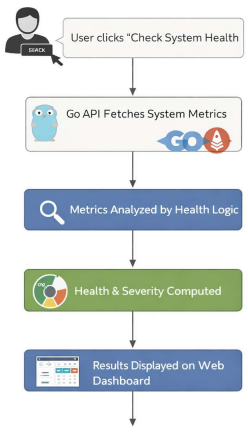

Fig. 1 User Guide

The figure 1 illustrates the workflow of the Intelligent On-Demand System Health

Monitoring system. When the user clicks the "Check System Health" option, the request is sent to the Golang API, which retrieves system metrics such as CPU usage, memory utilization, disk usage, and network traffic from the monitoring tools. These collected metrics are analysed by the system's health logic to evaluate the performance and condition of the system. Based on the analysis, the system computes the health status and severity level of the resources. Finally, the monitoring results are displayed on a web dashboard, allowing administrators to easily observe system performance and detect potential issues in real time.

4 EXPERIMENT

Hardware requirements:
Processor: Intel core I3 / Pentium iv or above
Ram: 4 GB minimum
Hard Disk: 50 GB free space
Monitor: 14-inch or above
Input devices: Standard Keyboard and Mouse
Software requirements:
Operating system: Windows 10/Windows 11/ Linux
Programming Language: Golang
Front End: HTML, CSS, Javascript
Backend Framework: go http server
Monitoring Tool: prometheus node exporter
Hosting Platform: firebase
Browser: Google Chrome / Microsoft Edge

The system was tested under various load conditions including idle state, CPU stress, and memory pressure. During idle conditions, the system reported high health percentages and normal severity classification. Under CPU-intensive workloads, the system detected increased utilization and reduced the health score accordingly.

Across all scenarios, the API response time remained low and no background overhead was observed. The results confirm that the on-demand monitoring architecture is efficient, responsive, and reliable.

5 RESULT

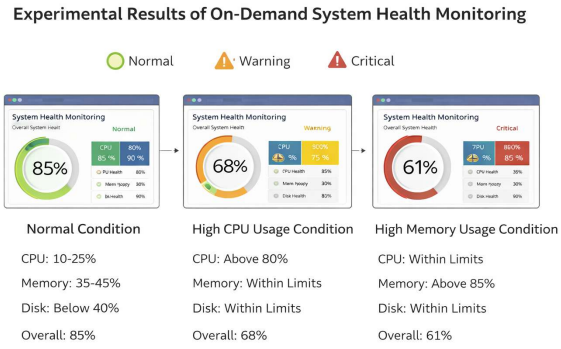

Fig. 2 Experimental Results

The experimental results demonstrate the effectiveness of the Intelligent On-Demand System Health Monitoring system in evaluating real-time hardware performance. The system calculates health percentages based on CPU usage, memory utilization, and disk activity, and determines an overall system health score along with a severity classification. The evaluation was conducted under different system load conditions to verify the accuracy of monitoring and severity detection.

A. Result Under Normal System Condition

Under normal conditions, system resources operated within safe limits. CPU utilization ranged between 10–25%, memory usage between 35–45%, and disk usage remained below 40%. After the health check was triggered, the backend retrieved metrics from Prometheus Node Exporter and applied predefined threshold logic. The system produced a high overall health score of approximately 85%, and the severity level was classified as Normal. The dashboard displayed the monitoring results clearly, with an API response time below 400 ms, confirming reliable monitoring performance.

B. Warning Condition

The warning condition was tested by generating high CPU load through multiple processes. In this scenario, CPU utilization exceeded 80% while memory and disk usage remained normal. The monitoring system detected the abnormal processor usage and reduced the CPU health score. The overall system health decreased to about 68%, and the severity level was classified as Warning. The dashboard reflected the updated health status, and

the response time remained within 500 ms, demonstrating effective detection of performance stress.

C. Critical Condition

The critical condition was simulated by running memory-intensive applications, causing memory utilization to exceed 85%. After initiating the health check, the system retrieved updated metrics and recalculated the health score. The overall system health dropped to approximately 61%, and the severity level was classified as Critical. The dashboard displayed the critical status immediately, indicating serious resource pressure. The monitoring system remained stable during this condition, confirming its ability to detect and classify severe system states accurately.

6 Future Enhancement

In the future, the proposed Intelligent On-Demand System Health Monitoring system can be enhanced by integrating advanced features such as automated alert notifications and predictive analysis. Machine learning techniques can be applied to analyze historical system metrics and predict potential failures before they occur. The system can also be extended to support monitoring of distributed and cloud-based infrastructures. Additionally, integrating visualization tools and mobile notification systems can further improve real-time monitoring and system management capabilities for administrators.

CONCLUSION

This paper presented an Intelligent On-Demand System Health Monitoring framework using Golang and Prometheus Node Exporter to monitor system performance efficiently. The system collects important hardware metrics such as CPU usage, memory utilization, disk activity, and network traffic in real time. By using an on-demand monitoring approach, the system reduces unnecessary resource consumption compared with traditional continuous monitoring systems. The experimental results demonstrated that the proposed framework can accurately evaluate system health and classify severity levels under different load conditions. The monitoring dashboard provides administrators with clear insights into system performance, enabling faster detection of potential issues. Overall, the proposed system offers a reliable and scalable solution for modern system health monitoring.

ACKNOWLEDGEMENT

The authors would like to express their sincere gratitude to the faculty members of the Department of Computer Science, Government Arts and Science College, Arakkonam, for their valuable guidance and continuous support during the development of this project. I also thank the authors and the institution for providing the necessary facilities and resources to successfully to complete the research work on Intelligent On-Demand System Health Monitoring Using Golang and Prometheus Node Exporter.

REFERENCES

[1] Kumar, R., & Ghosh, A., "Chaos Engineering For Resilient Microservice Systems," IEEE Transactions On Services Computing, 2021.
[2] Ali, S., et al., "A Survey of Chaos Engineering in Microservice Architectures," ACM Transactions on Software Engineering Methodology, 2021.
[3] Chisnall, D., "Enhancing Cloud-Native Application Reliability through Chaos Engineering," Springer Computing, 2020.
[4] Gupta, M., et al., "Microservices and Resilience: A Chaos Engineering Approach," Elsevier Future Generation Computer Systems, 2019.
[5] Espinoza, C., et al., "Chaos Engineering: From Theory to Practice in Microservice Architectures," IEEE Cloud Computing, 2022
[6] Diaz, S., et al., "Resilient Cloud Architectures Using Fault Injection," ACM SIGSOFT Software Engineering Notes, 2022.
[7] Wang, X., & Sun, Y., "Exploring Microservice Stability through Chaos Engineering," IEEE Access, 2021.
[8] Shah, K., et al., "Fault Injection for Cloud-Native Applications," ACM International Conference on Cloud Computing, 2020.
[9] Silva, L., & Pallis, G., "Survey on Chaos Engineering in Distributed Systems," ACM Computing Surveys, 2022.
[10] Basiri, A., et al., "Exploring Failure Injection Techniques for Distributed Systems," IEEE Transactions on Cloud Computing, 2020.

