



International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)





An Automated and Scalable Server Health Monitoring System

Dr. Thilagavathy A¹, Koyi Vishnu Vardhan Reddy², Mannepalli Sasi Kumar³, Moses Praneeth raj⁴

Associate Professor, Department of Science and Engineering, R.M.K. Engineering College, Chennai,
Tamil Nadu, India¹

UG Students, Department of Science and Engineering, R.M.K. Engineering College, Chennai, Tamil Nadu, India^{2,3,4}

ABSTRACT: Automation is critical for keeping system stability and performance across vast networks and server infrastructures in today's IT environments. Manual monitoring of server resources (CPU, memory, disk usage etc.) is time consuming and prone to human errors. In order to relief these issues, this project aims to implement an automated System Health Monitoring framework with Python, Prometheus and Grafana. An automation script implemented in Python connects via SSH to multiple servers, gathers live system metrics, parses them, and exposes them to Prometheus for continuous monitoring. These metrics are visualized and explored via Grafana, which gives the systems team immediate insight into how well the system is performing as well as what resources are being consumed. It helps in early failure detection, helps with operational efficiency & reduces manual monitoring tasks. Such an automation approach incorporates high reliability, scalability and improved visibility across multi-vendor environments (Linux, AIX and Solaris servers).

KEYWORDS: System Health Monitoring, Python Automation, Prometheus, Grafana, SSH Monitoring, Server Performance, Distributed Systems.

I. INTRODUCTION

Today's enterprise IT infrastructures consist of broad ranges of server platforms and operating systems, including environments such as Linux, AIX, and Solaris. An important operational requirement is to ensure that such systems are always available and performing at their best. But traditional system health monitoring techniques rely heavily on manual intervention, as administrators log into specific servers and check CPU, memory, disk and running process utilization. This approach does not only lead to lags, inconsistencies, and high operational overhead; it also is not a very scalable solution. Hundreds of servers in an infrastructure make manual health checks impossible and not proactive but reactive. Challenges such as delayed fault detection, inconsistent command outputs across disparate operating systems and a lack of central visibility can lead to increasing down time and decreasing reliability of the service. The only solution to overcome these challenges is an Automated, Centralized, Scalable monitoring system which can work in all types of server environments.

The automated server health monitoring framework presented in this paper uses Prometheus for metric aggregation and alerting, Grafana for visualization, and Python for SSH-based metric collection. The solution reduces manual monitoring efforts by about 80% and offers real-time insights into system performance while integrating seamlessly into enterprise environments.

II. RELATED WORKS

Early methods of system monitoring relied on scheduled cron jobs, manual inspection, and simple shell scripting. Although these techniques were more efficient than completely manual checks, they lacked real-time alerting capabilities, scalability, and centralized reporting, particularly in heterogeneous environments with Solaris, AIX, and Linux systems.

Although they introduced visualization and alerting features, centralized monitoring platforms like Nagios, Zabbix, and Datadog frequently have complicated configurations, expensive licensing, or few customization options. Prometheus's adaptable data model, pull-based architecture, and native alerting support have made it a popular choice in light of recent



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

developments in open-source monitoring tools. By offering sophisticated dashboards for historical trend analysis and real-time visualization, Grafana enhances Prometheus.

Because of its vast ecosystem and libraries, including the Prometheus client library for metric exposure and Paramiko for SSH connectivity, Python has become a dominant language for infrastructure automation. Together, these technologies make it possible to create scalable, affordable, and adaptable monitoring systems that are appropriate for contemporary business settings.

III. PROPOSED SOLUTION

To address this challenge, this work presents an automated monitoring system implemented in Python, Prometheus, and Grafana. The developed solution makes use of a Python-based script to establish secure SSH connections with multiple servers and collect key system health metrics like CPU utilization, memory usage, disk space consumption, and the number of active running processes. As this monitored environment consists of heterogeneous operating systems, the command outputs collected are parsed and standardized to make sure that all metric representations come out uniformly. Processed metrics will then be exposed through a Prometheus-compatible endpoint and are kept updated at regular intervals inside a time-series database. Prometheus will keep assessing these metrics against configured threshold values and send alerts in case abnormalities occur--for example, excessive CPU usage. Grafana helps in retrieving stored metrics from Prometheus, which are finally visualized in interactive dashboards to facilitate real-time monitoring, detailed analysis of historical trends, as well as thorough analysis of server activity in detail. The proposed model has been created to ensure it can be scaled to accommodate more servers seamlessly without compromising its performance in any way. The model will be able to minimize monitoring activities by approximately 80 percent by employing a health check process along with a measure to include timeouts as well.

IV. TECHNICAL CONVENTIONS

This section outlines the conventions applied to the writing, notation, and the metrics used in this paper. The IEEE formatting template was applied after all technical content was completed to maximize the quality of the content. Standard abbreviations are defined at first use, and the metrics for system performance (CPU, memory, and disk) are expressed in percentage ranges, derived from the utilized the operating system's reporting features. Presentation features such as manual pagination and numbering are managed by the IEEE template to keep everything uniform.

A. Abbreviations and Acronyms

Every acronym and abbreviation used in this work is defined at the point where it appears. For readers who are not familiar with the terminology used in distributed infrastructures and system monitoring, this approach guarantees clarity. The following are important acronyms used in this paper:

A cryptographic network protocol called SSH (Secure Shell) is used to provide safe remote server access.

The central processing unit, or CPU, is a server system's main processing component.

Random Access Memory, or RAM, is volatile memory used to run active processes.

An interface that facilitates communication between software components is called an API (Application Programming Interface).

The operating system, or OS, is the software that controls the hardware and software resources.

IBM's UNIX-based operating system is called AIX (Advanced Interactive Executive).

B. Units

The International System of Units, or SI, is the primary unit used in this paper to represent all measurable quantities associated with system performance.

Examples consist of:

- CPU utilization as a percentage (%)
- Gigabytes (GB) are used to express memory capacity.
- Gigabytes (GB) are used to measure disk usage.
- Time intervals measured in seconds (s)



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

To avoid confusion when interpreting metrics, mixed unit systems are avoided. Abbreviations and spelled-out units are not combined; instead, units are written consistently. For instance, disk capacity is not expressed in mixed forms like "gigabytes/GB," but rather as "gigabytes" in text and "GB" in figures.

To preserve numerical clarity, decimal values always contain a leading zero (e.g., 0.75, not .75). The system health metrics gathered from Linux, AIX, and Solaris servers are accurately interpreted thanks to these conventions.

C. Equations

The system health monitoring framework uses percentage-based CPU, memory, and disk resource utilization metrics to assess server performance. These metrics are used in Prometheus and Grafana for threshold-based alerting and visualization. They are derived from operating system-reported values.

The following is a definition of the utilization metrics:

$$\text{CPU usage} = \frac{\text{CPU used}}{\text{CPU total}} \times 100 \dots\dots\dots(1)$$

$$\text{Memory usage} = \frac{\text{Memory used}}{\text{Memory total}} \times 100 \dots\dots\dots(2)$$

$$\text{Disk usage} = \frac{\text{Disk used}}{\text{Disk total}} \times 100 \dots\dots\dots (3)$$

V. APPROACH AND FLOW

It describes the overall approach, methodology, and workflow to be followed for the design and implementation of the automated and scalable server health monitoring system. The approach puts much emphasis on the automation of metric collection, ensuring secure communication with real-time visualization and alerting across heterogeneous server environments.

A. Methodology

An Agile-based methodology was adopted for the development of the proposed system. The project was implemented in iterative phases, allowing continuous testing, validation, and refinement of features.

The methodology involves:

Identifying system health metrics such as CPU utilization, memory usage, and disk consumption.

Establishing secure SSH-based connections to Linux, AIX, and Solaris servers using Python.

Collecting and normalizing system metrics across different operating systems.

Exposing the collected metrics through a Prometheus-compatible /metrics endpoint.

Storing and evaluating metrics using Prometheus with threshold-based alerting.

Visualizing real-time and historical metrics using Grafana dashboards.

This structured approach ensures scalability, reliability, and minimal manual intervention in monitoring large server infrastructures.

B. Workflow

The workflow of the proposed system health monitoring method can be explained as depicted below: Initially, the server details are added/configured with the centralized configuration file. As such, the dynamic inclusion of Linux, AIX, and Solaris hosts can be made.

Once the host is set up, the monitoring system will now establish connections to each server securely through SSH authentication. It will be able to use various authentication methods, including password and key authentication. After establishing a connection successfully, health metrics are collected from servers. This involves obtaining critical health metrics like CPU, memory, disk, and active processes. In this case, these metrics are retrieved from servers. However, to make such metrics uniform across various operating systems, the command-line outputs are processed.

The normalized metrics are then made accessible through the /metrics endpoint, implemented in Prometheus-compatible fashion, by the exporter that is based on the Python code. The metrics are periodically scraped by Prometheus, making use of their time series database.



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Grafana queries for the stored metrics from Prometheus using PromQL and visualizes them through its interactive dashboards, which provide real-time visualization of server health and performance trends.

During this phase, the system checks whether predefined threshold values have been breached. An alert notification will be sent in case any metric crosses the configured limits; otherwise, if there's no threshold breach, then the system will continue with the monitoring cycle without an interruption.

This workflow provides a continuous, automated, and scalable monitoring of server health with proactive alerting and centralized visualization.

C. Tables

TABLE I. TECHNOLOGY STACK USED

| COMPONENT | TECHNOLOGY USED | PURPOSE |
|-------------------|-----------------|----------------------------------|
| MONITORING SCRIPT | PYTHON 3 | METRIC COLLECTION AND PARSING |
| METRIC STORAGE | PROMETHEUS | TIME-SERIES STORAGE AND ALTERING |
| VISUALIZATION | GRAFANA | DASHBOARDS AND TREND ANALYSIS |
| COMMUNICATION | SSH | SECURE REMOTE ACCESS |
| DEPLOYMENT | DOCKER COMPOSE | SERVICE ORCHESTRATION |

TABLE II. SUPPORTED OPERATING SYSTEM

| OPERATING SYSTEM | VERSION TYPE | MONITORING METHOD |
|------------------|---------------|--------------------|
| LINUX | RHEL / UBUNTU | SSH-BASED COMMANDS |
| AIX | IBM UNIX | SSH-BASED COMMANDS |
| SOLARIS | ORACLE UNIX | SSH-BASED COMMANDS |

TABLE III. MANUAL VS AUTOMATED MONITORING

| ASPECT | MANUAL MONITORING | AUTOMATED MONITORING |
|----------------------|-------------------|----------------------|
| EFFORT REQUIRED | HIGH | LOW |
| MONITORING FREQUENCY | PERIODIC | CONTINUOUS |
| SCALABILITY | LIMITED | HIGH |
| ERROR PROBABILITY | HIGH | LOW |
| RESPONSE TIME | SLOW | FAST |
| TABLE IV. | TABLE V. | TABLE VI. |



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

D. Architecture Diagram

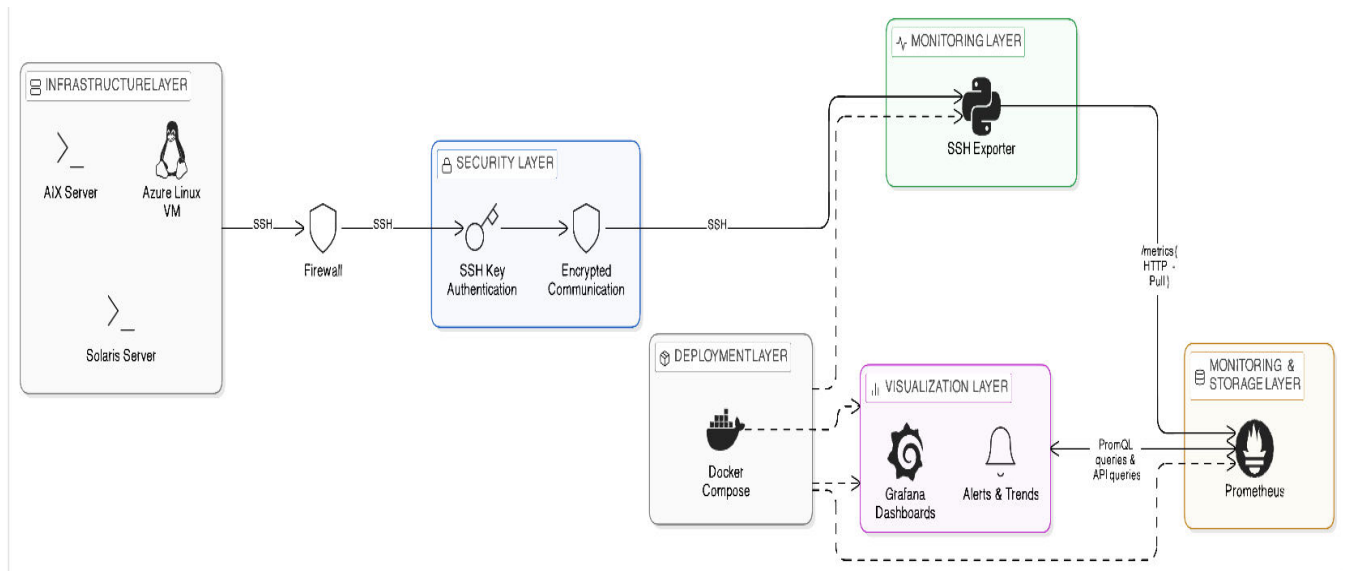


Fig.1 Architecture Diagram

The proposed system's architecture is modular and safe because it is built in layers. There is an infrastructure layer with different types of servers, a security layer that lets SSH encrypted communication, a monitoring layer that uses a Python exporter, and a layer that focuses on monitoring and storing information through Prometheus. Docker Compose is the layer that handles deployment, and Grafana is the layer that handles visualization. It makes it possible to watch servers in different places from one central location.

E. Libraries used

- Paramiko: This Python module is essential for making SSH connections to remote servers. It securely executes commands on those servers and helps gather system data efficiently.
- time: Used to handle timing operations like pausing the program briefly, tracking elapsed time, or setting up scheduled checks at regular intervals.
- os: Provides tools for interacting with the underlying operating system, such as verifying available disk space, managing folders for storing logs, and working with file paths dynamically.
- sys: Grants access to system-specific variables and command-line inputs, which makes it easier to supply server lists or runtime parameters to the monitoring script.
- paramiko_expect (SSHClientInteraction): Facilitates interactive SSH sessions by allowing commands that require ongoing interaction (like top or tail commands) to run and capture their output automatically.
- glob: Helps fetch lists of files from directories based on patterns, which is handy for collecting multiple logs across servers for further analysis.
- multiprocessing (Process): Allows the program to launch multiple monitoring tasks simultaneously, so several servers can be checked in parallel without causing delays.
- prometheus_client: Enables the Python script to expose metrics in a format that Prometheus can scrape, store, and then visualize through Grafana dashboards.
- json: Used for formatting collected system information and metrics into JSON, making the data structured and easy for other services like Prometheus to consume.



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

VI. OUPUTS AND DIAGRAM

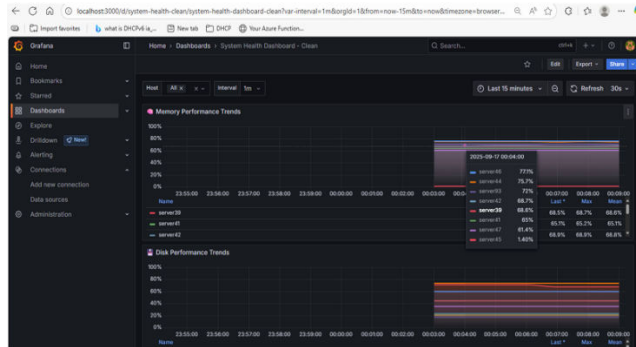


Fig 2. Grafana visual-1

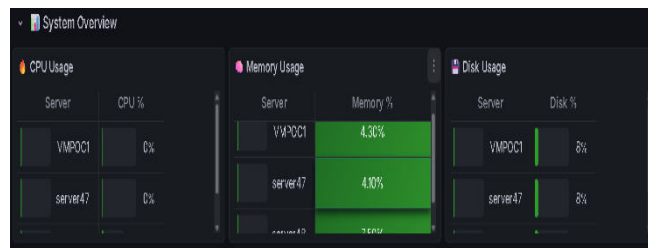


Fig 3. Grafana visual-2



Fig 4. Grafana visual-3



Fig 5. Grafana visual-4



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

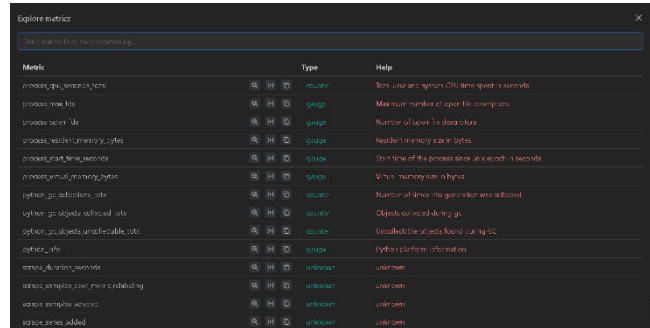


Fig 6 Prometheus Dashboard 1

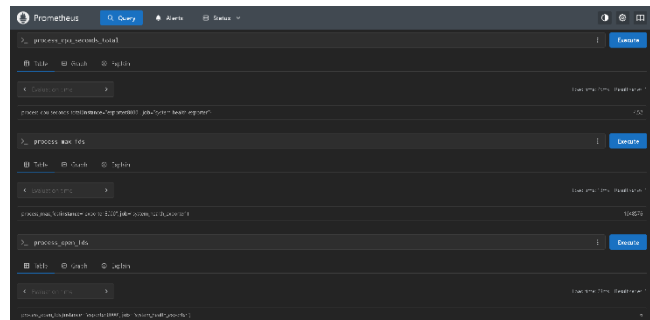


Fig 7 Prometheus Dashboard 2

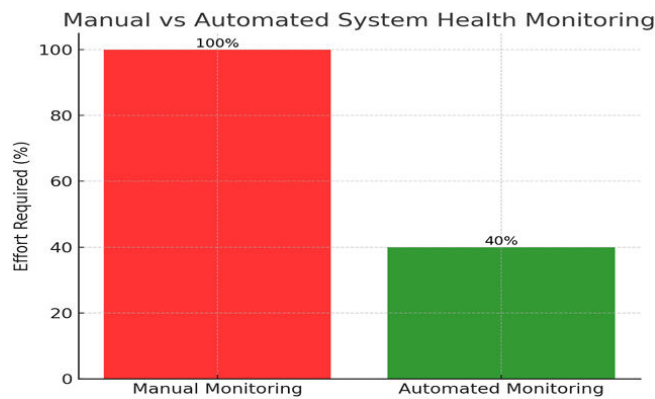


Fig.8 Manual vs Automated System monitoring

VII. CONCLUSION

It talks about the idea of using Python, Prometheus, and Grafana to make an automated system health monitoring solution for server environments that are different from each other. It lets you safely and continuously collect CPU, memory, and disk metrics from the systems while also giving you a central view and proactive alerts. The solution replaces manual health checks, which makes it easier to scale, more reliable, and more efficient to run. It also lets you monitor Linux, AIX, and Solaris servers in real time.

VIII. LITERATURE SURVEY

System health monitoring The area has been well studied. All large IT systems in business and government have to keep up their availability and performance. Traditionally, administrators would get into each server by the root or privileged



International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

user login and run some simple commands such as 'ps' (process status), to find out what CPU, memory and disk is being used. The disadvantage of solitary man operation combined with historical checking method is that it inevitably leads to an expanded possibility for failure in all those areas. Very rare or even nonexistent, therefore, must be one system which unattended that ingenuity itself became too much work exerted on behalf of machines. Early attempts to automate monitoring and usage employed shell scripts with cron jobs. Although these routines improved efficiency, they provided both conventional reports in a clumsy manner. We had no real-time statistics, alerting capabilities across poly-OS environments such as Linux, AIX, and Solaris. Newer solutions are built on stacks that leverage central monitoring systems like Nagios, Zabbix and Datadog that used for visualization and alerting. But these are often quite complicated to configure, may have associated licensing costs or lack the ability to customize as needed. Recently, Prometheus has emerged as a popular option for an open-source, time-series database specialized in metrics monitoring. It has a flexible data model, an alerting system built in, and works with custom exporters, which makes it a good choice. Grafana also offers rich, interactive dashboards that make it easier to keep an eye on things and fix problems. Python is also a popular choice for making custom monitoring tools because it is flexible and has a lot of libraries and frameworks. Libraries like Paramiko (for SSH connections), Regex (for parsing command outputs), and the Prometheus client library (for metric exposure) make it easier to build strong, automated monitoring solutions.

REFERENCES

1. **Mohammed Daffalla Elradi**, Prometheus and Grafana: A Metrics-Focused Monitoring Stack, *Journal of Computer Allied Intelligence*, Vol. 3, No. 3, 2025.
2. **Pragathi B. C., Hrithik Maddirala, Sneha M.**, Server Performance Monitoring Using Prometheus and Grafana, *International Journal of Computer Applications (IJCA)*, 2024.
3. **Heli Barrett, Jere Matthews, Aino Ford**, Observability and Metrics-Based Monitoring in Cloud Systems, *Springer Journal of Cloud Computing*, 2022.
4. **Harold Castro**, Design and Implementation of Scalable Monitoring Systems Using Prometheus, *IEEE Access*, 2021.
5. **Ejielo Ogbuefi, Oyejide Timothy Odofin**, Secure and Efficient Remote System Monitoring in Cloud Environments, *Elsevier Future Generation Computer Systems*, 2022.
6. **Abraham Ayodeji Abayomi**, Agentless Infrastructure Monitoring Using Open-Source Tools, *International Journal of Engineering Research and Technology (IJERT)*, 2023.
7. **James Turnbull, Monitoring with Prometheus, O'Reilly Media**, 2018. Tiia Leppänen, Monitoring and Visualization Using Prometheus and Grafana, Bachelor's Thesis, 2021.
8. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., and Wilkes, J., "Borg, Omega, and Kubernetes," *Communications of the ACM*, Vol. 59, No. 5, pp. 50–57, 2016.
9. Turnbull, J., *The Art of Monitoring*, Turnbull Press, 2016.
10. Sigelman, B. H., et al., "Dapper, a Large-Scale Distributed Systems Tracing Infrastructure," *Google Research*, 2010.
11. Pahl, C., "Containerization and the PaaS Cloud," *IEEE Cloud Computing*, Vol. 2, No. 3, pp. 24–31, 2015.
12. Zabbix LLC, "Zabbix Monitoring Solution Architecture and Performance Analysis," *White Paper*, 2021.
13. Nguyen, T. T., Kim, S., and Park, J., "Performance Monitoring and Alerting in Distributed Systems Using Prometheus," *International Journal of Distributed Sensor Networks*, 2020.
14. IBM Corporation, "AIX Performance Monitoring and Tuning Guide," *IBM Documentation*, 2022.
15. Oracle Corporation, "Solaris System Performance Monitoring and Analysis," *Oracle Technical White Paper*, 2021.
16. Behl, A., and Behl, K., *Cybersecurity and Cyberwar: What Everyone Needs to Know*, Oxford University Press, 2017.
17. Paramiko Developers, "Paramiko: Python SSHv2 Protocol Library," *Official Documentation*, 2024.
18. Prometheus Authors, "Prometheus Monitoring System Documentation," *The Linux Foundation*, 2024.
19. Grafana Labs, "Grafana Observability Platform Documentation," *Grafana Labs*, 2024.
20. Docker Inc., "Docker Compose: Defining and Running Multi-Container Applications," *Docker Documentation*, 2023.
21. Kleppmann, M., *Designing Data-Intensive Applications*, O'Reilly Media, 2017.



INTERNATIONAL
STANDARD
SERIAL
NUMBER
INDIA



INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

 9940 572 462  6381 907 438  ijircce@gmail.com



www.ijircce.com

Scan to save the contact details