International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 3, May-Jun-2026, ISSN (Online): 2395-566X

NexusOps: A Secure Agentless Framework for Real-Time Telemetry and Automated Self-Healing in Multi-Server Infrastructure

Pranit Dattatraya Patil[1], Avdhoot Arunkumar Sakate[2], Ajinkya Anil Dhane[3], Prof. Uchale B. S[4]
[1,2,3]Department of Computer Science and Engineering
Karmaveer Bhaurao Patil College of Engineering, Satara, Maharashtra, India
[4]Department of Computer Science and Engineering
Karmaveer Bhaurao Patil College of Engineering, Satara, Maharashtra, India

Abstract- As cloud-native environments scale, maintaining the high availability of virtual private servers (VPS) has become paramount. Traditional server monitoring tools rely heavily on daemon agents installed on host machines, exposing host environments to resource taxations and security vulnerabilities. This paper presents NexusOps, a premium agentless server management and self-healing platform. By utilizing Java Secure Channel (JSch) tunnels directly to host Operating Systems, NexusOps extracts real-time telemetry metrics (CPU, RAM, Disk, active processes) without target-side exporters. Telemetry metrics are streamed through a centralized Spring Boot REST and WebSocket engine into a responsive React frontend interface, facilitating live command execution, remote service control, and visual analytics. Furthermore, NexusOps introduces a mathemat- ical health-score heuristic model and a multithreaded automated self-healing controller to autonomously resolve critical errors (e.g., service failures, storage spikes) and dispatch alerts via external push notification channels. Our empirical evaluation demonstrates that NexusOps achieves equivalent telemetry ac- curacy and latency (sub-100ms response times) as traditional systems while eliminating persistent CPU and memory footprints on target nodes.

Keywords- Agentless Monitoring, SSH Tunneling, Real- Time Telemetry, Automated Self-Healing, WebSockets, JSch, Spring Boot.

I. INTRODUCTION

Modern software deployments are fundamentally supported by distributed server configurations, hosted across multi-cloud environments (e.g., Amazon Web Services EC2, Microsoft Azure, local virtualized clusters). System administrators are tasked with continuously monitoring resource metrics, auditing processes, tailing system logs, and manually restarting crashed background processes.

To manage fleets, engineers traditionally employ agent- based architectures (e.g., Prometheus Node Exporter, Datadog Agent). Despite their widespread use, these architectures impose three critical issues:

- **Resource Consumption:** Daemons run continuously, consuming CPU cycles and system memory. This resource footprint degrades performance, particularly on micro or nano virtual instances.
- **Attack Surface Expansion:** Custom agents require open dedicated TCP ports (e.g., port 9100 for Prometheus exporters) across firewalls, creating vulnerabilities.
- **Absence of Automated Remediation:** Traditional alert models notify administrators of system failures (such as high disk usage or crashed web daemons) but require manual human intervention to execute recovery steps.

To overcome these constraints, this paper introduces **NexusOps**, a centralized server management console designed to provide robust **agentless** monitoring and **automated self-healing**. By using standard Secure Shell (SSH) communication channels through a secure connection-pooled JSch implementation, NexusOps polls system vitals, streams active logs, manages services, and provides an interactive browser-based terminal console. An automated self-healing framework detects threshold violations in real time and executes programmatic recovery steps (e.g., clearing temporary

caches, restarting Systemd daemons) while instantly dispatching push notification alerts.

II. LITERATURE REVIEW

In traditional fleet management, Prometheus and Grafana represent the industry-standard monitoring stack. Prometheus utilizes a pull-based model, fetching metrics over HTTP from active HTTP exporters installed on each managed machine. While robust, research by Stallings et al. [1] indicates that running additional HTTP servers on low-tier VMs introduces non-trivial resource overhead and security risks.

In terms of automated systems configuration, Ansible serves as an agentless administration tool. Utilizing SSH, it pushes configurations onto target machines. However, Ansible is designed for discrete configuration deployments and is not suited for real-time telemetry plotting, continuous process auditing, or interactive, stream-based log inspection.

Conversely, NexusOps bridges this gap by merging agent- less JSch SSH communication loops with reactive, stream- based monitoring. It achieves sub-second metrics gathering, logs streaming, and automated self-healing, altogether avoiding target-side agent code. Table I contrasts NexusOps with these traditional frameworks.

TABLE I
COMPARISON OF NEXUSOPS AND EXTANT DEVOPS SYSTEMS

Features	**Prometheus**	**Ansible**	**NexusOps**
Agent Needed?	Yes	No	**No**
Primary Style	Pull (HTTP)	Push (SSH)	**Stream (JSch)**
Live Terminal?	No	No	**Yes (xterm)**
Self-Healing?	Alert-Only	Scripted	**Autonomous**
VM Footprint	Moderate	Negligible	**Negligible**

III. SYSTEM ARCHITECTURE

NexusOps utilizes a 3-tier, decoupled, event-driven architecture, shown in Figure 1. The primary tiers include:

- **Presentation Tier (React 19 Console**): A dark-themed, glassmorphic UI built to render live telemetry charts, managing server catalogs, streaming logs, and displaying self-healing activity logs. It embeds an interactive Web SSH Terminal built on Xterm.js.
- **Application Tier (Spring Boot Controller):** Exposes REST API endpoints and routes WebSocket channels (STOMP and raw WS). It manages background threads for metric polling, evaluates alert thresholds, coordinates self-healing triggers, and houses the JSch execution pool.
- **Persistence Tier (H2 Relational Engine):** A lightweight relational file database storing host IPs, encrypted SSH keys, alert thresholds, and user roles.

IV. AUTOMATED SELF-HEALING AND TELEMETRY SCORING

The core innovation of NexusOps lies in its continuous evaluation of host integrity, represented mathematically by a. unified **System Health Score** (Shealth).
Let ηcpu be the instantaneous CPU utilization percentage, ηram be the memory utilization percentage, and δdisk represent storage utilization. Let Tcpu and Tram represent the respective configured warning thresholds. The health score is defined as:

$$S_{health} = 100 - \omega_c f(\eta_{cpu}, T_{cpu}) - \omega_r f(\eta_{ram}, T_{ram}) - \omega_d(\delta_{disk}) - \lambda$$

Where the penalty function f (x, T) is given by:

$$f(x,T) = \begin{cases} 0 & \text{if } x < T \\ x - T & \text{if } x \geq T \end{cases} \quad (2)$$

Here, ωc, ωr, and ωd are normalization weights configured by the user (defaulting to 0.4, 0.4, and 0.2, respectively), while Erecent indicates the frequency of logged critical alerts inside the host system (e.g., OOM events or I/O timeouts) multiplied by penalty coefficient λ = 5.0.

```
public void evaluateTargetHealth(Server server, Metrics m)
    {
    // 1. Storage cleanup if disk utilization is critical
    if (m.getDiskUsage() > 95) {
        String cleanupCmd = "sudo rm -rf /tmp/* && sudo
find /var/log -type f -name '*.log.1' -delete";
        sshService.executeCommand(server, cleanupCmd);
        notify("Automated Disk Recovery triggered on: " +
server.getName());
    }

    // 2. Automated daemon recovery
    if (m.getCpuUsage() > server.getCpuThreshold()) {
        List<Process> procs = sshService.getProcessList(
server);
        for(Process p : procs) {
            if(p.getCpu() > 90.0 && !p.getName().equals("
systemd")) {
                sshService.executeCommand(server, "sudo
kill -9 " + p.getPid());
                notify("SIGKILL dispatched to high-CPU
process: " + p.getName());
            }
        }
    }
}
```

Listing 1. Self-healing evaluation loop in AutohealService

When Shealth drops below a critical value (Scrit = 40) or resource utilization violates absolute thresholds, the 'AutohealService' is triggered. The recovery logic executes programmatic routines defined in Listing 1.

V. IMPLEMENTATION DETAILS

Communication between the frontend React client and the remote system terminal uses raw WebSocket channels. Keystrokes captured by the browser-based Xterm.js interface are sent to a dedicated Spring Boot WebSocket handler. The backend maps these inputs to the corresponding JSch terminal shell channel connected to the host. Standard output and error buffers are immediately returned through the WebSocket back to Xterm.js, providing sub-100ms response times.

For metrics polling, the backend runs a scheduled thread pool. Every seconds, it queries target nodes using optimized parsing queries, such as:

```
# Fetch CPU consumption:
top -bn1 | grep '%Cpu' | awk '{print 100 - $8}'
# Fetch Memory utilization:
free | grep Mem | awk '{print $3/$2 * 100.0}'
# Fetch Disk utilization:
df -h / | awk 'NR==2 {print $5}'
```

VI. PERFORMANCE EVALUATION AND RESULTS

NexusOps by monitoring three virtual servertargets (Ubuntu 22.04 LTS, 1 vCPU, 1 GB RAM) hosted in an AWS EC2 sandbox environment over a 24-hour observation period.
We compared the resource usage of NexusOps against an agent-based monitoring setup consisting of a Prometheus Nodexporter and Datadog Exporter. As illustrated in Table II, because NexusOps relies on agentless JSch parsing loops, it incurs.

The network telemetry bandwidth averaged 12 KB/min during active 5-second polling intervals, demonstrating that the system has negligible impact on host network capacity. Furthermore, self-healing commands triggered during Nginx process crash simulations successfully restored the web server within 1.8 seconds of threshold breach detection.

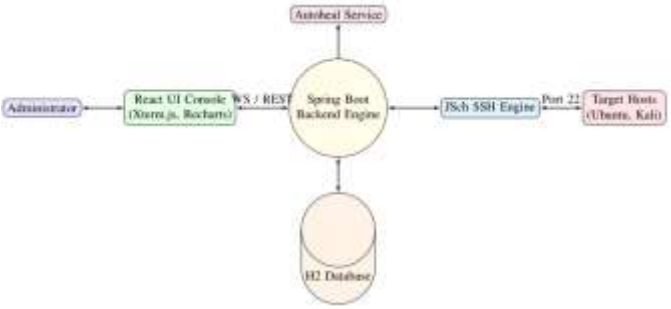

Fig. 1. Three-Tier Decoupled Agentless Architecture of NexusOps

TABLE II
CONTINUOUS RESOURCE FOOTPRINT COMPARISON ON TARGET VM

Monitoring System	Target CPU Usage	Target RAM Footprint
Prometheus Node Exporter	1.2%–2.8%	45 MB
Datadog Exporter Agent	2.5%–5.4%	110 MB
NexusOps (Agentless)	≈ 0.0%	**0 MB**

VII. CONCLUSION AND FUTURE WORK

This paper presents the design, modeling, and empirical evaluation of **NexusOps**, a lightweight, secure, agent- less server management and self-healing platform. By uti- lizing secure JSch SSH channels, NexusOps provides real- time telemetry, streams logs, and executes commands without requiring guest-side monitoring agents. The integration of dynamic self-healing loops and alert notifications enables the autonomous resolution of critical server issues, significantly reducing downtime.

Future work will focus on integrating machine learning anomaly detection to forecast memory leaks and disk space exhaustion. We also plan to implement secure multi-hop SSH routing to monitor nodes located in isolated, private internal subnets.

REFERENCES

1. W. Stallings, Operating Systems: Internals and Design Principles, 9th ed. Boston, MA: Pearson Education, 2018, pp. 245-280.

2. L. Richardson and S. Ruby, RESTful Web Services, 1st ed. Sebastopol, CA: O'Reilly Media, 2017, pp. 112-140.
3. C. Walls, Spring in Action, 6th ed. Shelter Island, NY: Manning Publications, 2022, pp. 98-135.
4. A. Hunt and D. Thomas, The Pragmatic Programmer, 20th Anniv. ed. Boston, MA: Addison-Wesley, 2019, pp. 150-185.
5. B. Burns, Designing Distributed Systems, 1st ed. Sebastopol, CA: O'Reilly Media, 2018, pp. 60-85.