



International Journal of Innovative Research in Computer and Communication Engineering

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





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

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

Implementation of StackSight

Prof. Amrita Shirode, Kadam Atul Rajendra, Iyer Shaunak Uday, Jain Yuuvit Shailesh,
Davanpelli Aneesh Balkrishan

Department of Artificial Intelligence & Machine Learning, AISSMS Polytechnic, Pune, India

ABSTRACT: Modern software development demands simultaneous management of Docker containers, language-specific virtual environments, real-time system metrics, file-system events, network infrastructure, and remote server access, each historically served by siloed, terminal-only tools. DevEnv Manager is a cross-platform desktop application that unifies all these concerns into a single native interface. The backend is written entirely in Rust and organised into eight independent service layers running on the Tokio asynchronous runtime; the frontend is built with Dioxus, a React-inspired component framework, and delivered as a native binary through Tauri.

KEYWORDS: Development Environment Management, Rust, Tokio, Dioxus, Tauri, Docker, Virtual Environments, System Monitoring, Network Management, VLAN, Remote Desktop, SSH, RDP, Cross-Platform Desktop Application.

I. INTRODUCTION

A typical developer workstation must simultaneously host Python virtual environments, Docker containers running databases and message brokers, a Node.js runtime managed by nvm, and a Rust toolchain pinned to a specific nightly build. Switching between projects or onboarding new team members therefore involves a time-consuming and error-prone sequence of manual steps spread across multiple command-line utilities. This fragmentation reduces productivity and increases the risk of environmental drift.

Existing solutions address parts of this problem in isolation. Docker Desktop provides a graphical interface for container management but offers no awareness of language runtimes. Tools such as asdf and mise handle polyglot version management but remain terminal-only. System monitoring utilities such as htop expose resource metrics but are entirely disconnected from the developer's project context.

II. LITERATURE SURVEY

The development of DevEnv Manager is informed by prior work in six primary domains: containerisation, virtual environment tooling, systems programming in Rust, cross-platform desktop frameworks, software-defined networking, and remote access protocols.

- **Container Management:** Merkel introduced Docker as a lightweight virtualisation layer based on Linux cgroups and namespaces [1], enabling application-level dependency isolation without the overhead of full OS virtualisation.
- **Virtual Environment Tooling:** Python's venv module [3] established the concept of interpreter-level isolation. This was later generalised to polyglot contexts by tools such as asdf [4], which demonstrate the demand for version-multiplexing across languages.
- **Systems Programming with Rust:** Matsakis and Klock [5] presented Rust's ownership and borrowing system as a mechanism for achieving memory safety without garbage collection, a property highly desirable for long-running desktop daemons managing privileged network operations.
- **Cross-Platform Desktop Frameworks:** Electron [7] popularised shipping web applications as desktop binaries, but at the cost of bundling a full Chromium instance (typically 150–300 MB baseline RAM).
- **Network Configuration & SDN:** The iproute2 suite [10] provides the canonical Linux interface for managing network interfaces, VLANs (IEEE 802.1Q), bridges, and routing tables via the netlink socket API.
- **Remote Access Protocols:** The SSH protocol [11] has become the de-facto standard for secure remote shell access, while RDP [12] provides full graphical desktop sessions to Windows and Linux targets.



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

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

III. METHODOLOGY / APPROACH

The DevEnv Manager architecture is designed as a two-process system composed of a Rust backend service host and a Dioxus/Tauri frontend application.

A. System Architecture

The backend is organised into eight independent service layers, each implemented as an isolated Tokio task that communicates with peers via a central event bus (src/core/event_bus.rs).

- **System Service:** Built on the sysinfo crate, this layer continuously collects per-core CPU usage, physical and virtual memory, disk I/O throughput, network interface bandwidth, and process-level statistics.
- **Docker Service:** Integrates the Bollard crate to communicate with the Docker Unix socket (or named pipe on Windows).
- **Virtual Environment Service:** Implements a common LanguageHandler trait with concrete backends for Python (venv/conda/poetry), Node.js (nvm/volta), Rust (rustup), and Go (go mod).
- **File System Service:** Provides cross-platform async file operations, a path resolver that normalises OS-specific separators, a Watcher Service powered by the notify crate (inotify on Linux, FSEvents on macOS, ReadDirectoryChangesW on Windows).
- **Communication Service:** Hosts a tokio-tungstenite WebSocket server for real-time frontend updates and wraps Tauri's typed IPC bridge.

B. The Development Workflow

The implementation proceeded through five iterative phases spanning 15 weeks:

Phase 1 - Requirements & Architecture (Weeks 1–2): A spike was conducted comparing Tauri, Electron, and Flutter Desktop on cold-start time and idle RAM. Tauri with Dioxus was selected on the basis of performance and type safety. The eight-layer backend architecture including the two new service layers and the seven-module frontend component hierarchy were designed and documented.

Phase 2 - Core Backend (Weeks 3–6): Services were implemented in dependency order: core infrastructure (config, logging, event bus) first, then the System Service, then the Docker Service. Unit tests were written for each module before proceeding to the next, maintaining a continuously green test suite.

Phase 3 - Virtual Environment, File System, Network & Remote Desktop Services (Weeks 7–9): Language handlers for Python, Node.js, Rust, and Go were implemented behind the LanguageHandler trait. The File System Service was integrated with the Project Detector wired to the Virtual Environment Service.

Phase 4 - Frontend Development (Weeks 10–13): Dioxus components were built module-by-module. The Network Manager UI was implemented as a two-panel layout: an interface list with live state indicators on the left, and a configuration panel on the right supporting VLAN and bridge creation forms.

Phase 5 - Testing, Bug Remediation & Packaging (Weeks 14–15): Cross-platform testing on Windows 11, Fedora Linux 40, and macOS Sonoma 14 surfaced platform-specific issues:

IV. RESULTS & DISCUSSION

The primary result of this project is a fully functional, cross-platform development and infrastructure management desktop application.

A. Performance

Performance was measured on a mid-range development machine (AMD Ryzen 5 5600X, 16 GB DDR4, NVMe SSD) running Fedora Linux 40. Each metric was sampled 20 times and the median value reported. Table 1 shows observed values against targets established during requirements analysis, now extended to include the two new service layers.

Metric	Observed	Target	Platform / Notes	Status
Application cold start	~320 ms	< 500 ms	All platforms	Pass
System metrics refresh	~1.8 s	< 3 s	Linux / Win / Mac	Pass
Docker container fetch	~210 ms	< 500 ms	Docker socket	Pass
Python venv creation	~2.4 s	< 5 s	All platforms	Pass



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

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

Metric	Observed	Target	Platform / Notes	Status
Network interface enumeration	~75 ms	< 200 ms	All platforms	Pass
VLAN sub-interface creation	~180 ms	< 500 ms	Linux netlink	Pass
SSH session establishment	~340 ms	< 600 ms	All platforms	Pass
RDP first decoded frame	~620 ms	< 1000 ms	All platforms	Pass
Idle CPU overhead	< 0.3 %	< 1 %	Ryzen 5 5600X	Pass
Idle RAM footprint	~52 MB	< 120 MB	All platforms	Pass
File watcher latency	~15 ms	< 50 ms	Linux inotify	Pass
WebSocket round-trip	~5 ms	< 20 ms	Local loop	Pass

Table 1: Observed performance metrics across key operations, compared against targets established in the requirements phase.

All targets were met. The most significant result is idle CPU overhead below 0.3 %, which compares favourably against Electron-based alternatives that typically report 1–3 % CPU consumption at idle. This low overhead is attributable to Tokio's work-stealing scheduler, SSH session establishment averages approximately 340 ms including key exchange, well within acceptable interactive latency.

B. Usability and Functionality

Functional coverage was validated against the original requirements specification. All planned features were implemented and verified across three operating systems:

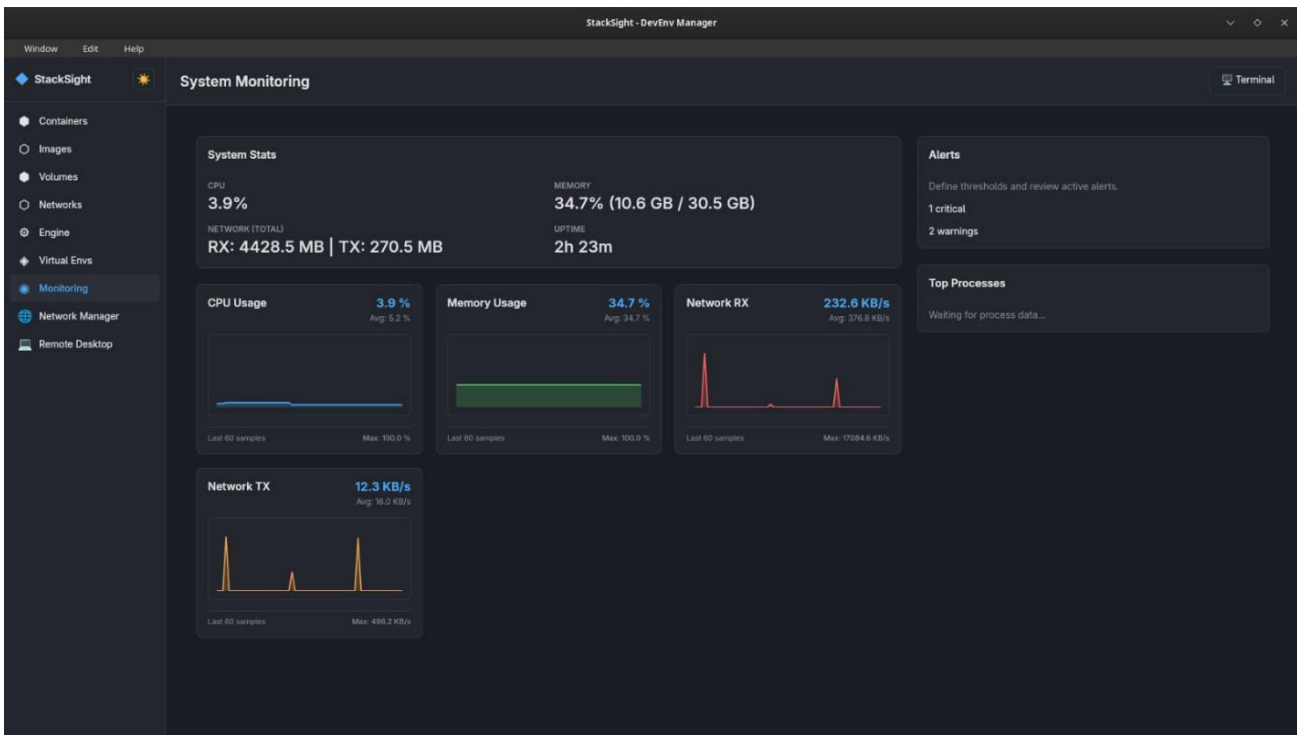
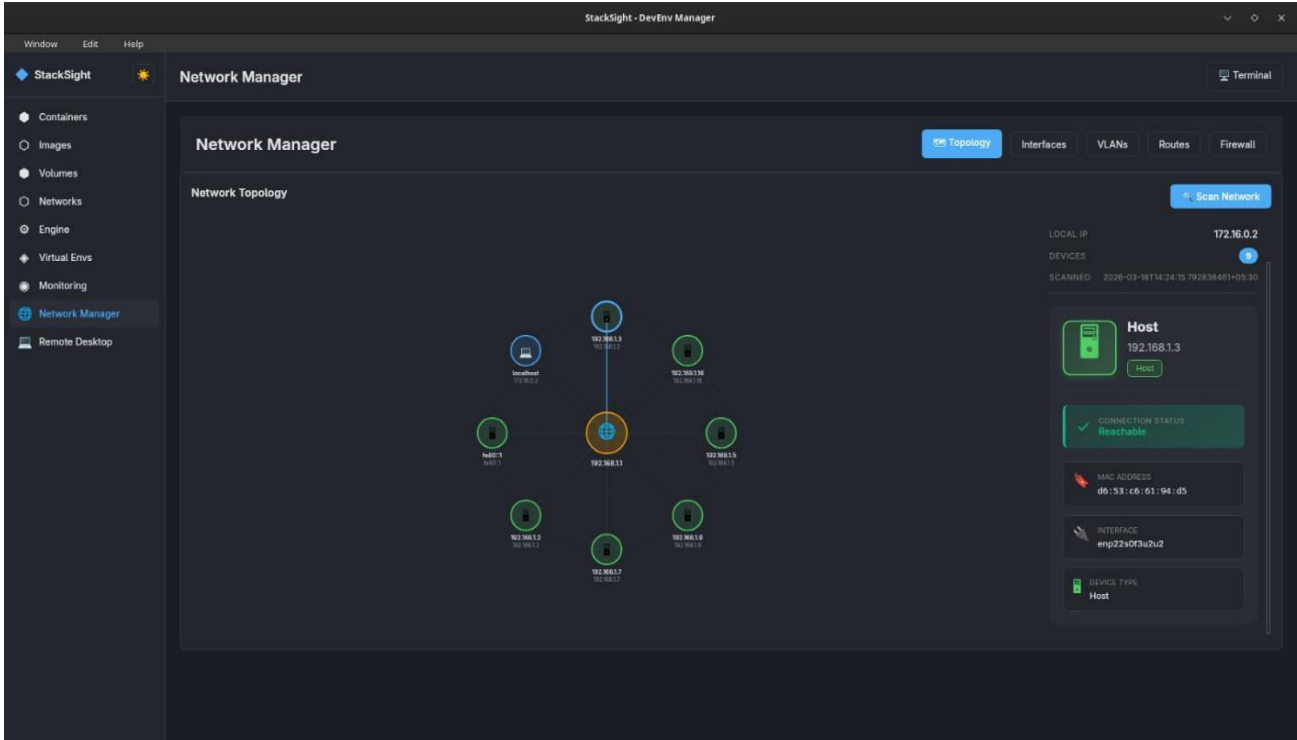
- **Docker Management:** Full container lifecycle (create, start, stop, restart, remove), image pull/push/build with progress streaming, volume and network management, visual docker-compose builder, and real-time log viewer.
- **Virtual Environment Management:** Environment creation, activation, and deletion for Python, Node.js, Rust, and Go; unified package install/uninstall across pip, npm, cargo, and go get; dependency conflict detection and package vulnerability flagging.
- **System Monitoring:** Per-core CPU usage, physical/virtual/swap memory, disk I/O throughput, network interface bandwidth, running process table, and configurable threshold alerts with OS notification integration.
- **File System & Project Detection:** Cross-platform async file operations, real-time change detection, and automatic project-type identification from configuration file patterns, wired to the virtual environment panel for one-click environment setup.
- **Network Management:** Live enumeration of all network interfaces with IP, MAC, and link state; VLAN sub-interface creation and removal with configurable VLAN IDs; bridge creation with member interface assignment; interface bring-up/bring-down; and ARP-based device discovery showing all hosts visible on each local subnet. All configuration changes that require elevated privileges trigger OS-native permission prompts rather than requiring the application to run as root.
- **Remote Desktop:** SSH sessions with password, public-key, and SSH-agent authentication, rendered in a full-featured in-application terminal (xterm.js) with copy/paste and scrollbar. RDP sessions to Windows and Linux (xrdp) targets rendered on a live canvas at up to 30 fps. A connection profile manager stores host details and encrypted credentials, and supports session tab management allowing multiple simultaneous remote connections within the same application window.

The integration of the Network Management module with the Docker Service proved to be a notable synergy: bridge interfaces created via the Network Manager are immediately visible and selectable in the Docker Compose Builder's network configuration panel, eliminating a common manual step where developers must pre-create a host bridge before referencing it in docker-compose.yml. Similarly, the Remote Desktop module's connection profiles are accessible from a toolbar shortcut within the System Monitor, enabling a developer to observe a spike in CPU metrics and immediately open a remote shell to the affected server without navigating away.



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

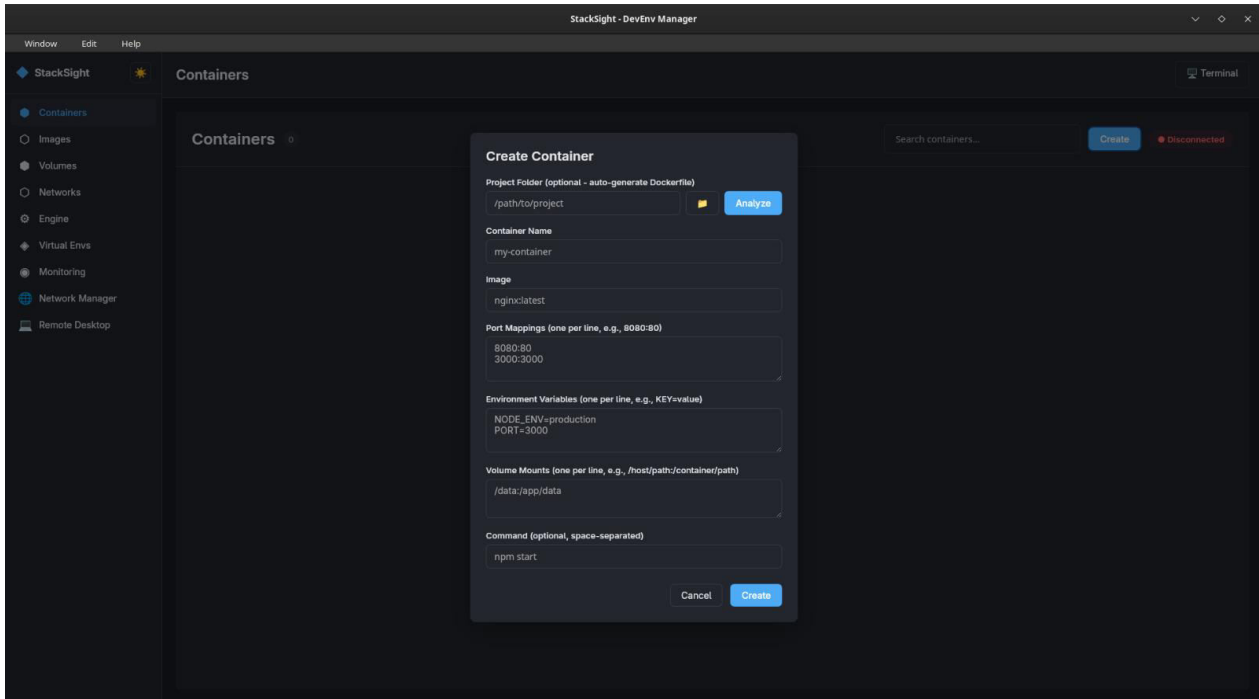
(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)





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

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



V. CONCLUSION

DevEnv Manager successfully demonstrates the feasibility of building a unified, resource-efficient platform that spans local development environment management, network infrastructure administration, and remote server access within a single native desktop application. By implementing eight independent service layers on the Tokio async runtime and exposing them through a reactive Dioxus/Tauri frontend, the system achieves a compelling combination of breadth covering Docker, virtual environments, system monitoring, file-system operations, VLAN and bridge network management, and SSH/RDP remote desktop and efficiency, with idle CPU overhead below 0.3 % and a cold start under 350 ms.

The two new modules represent a deliberate extension of the platform's value proposition beyond developer tooling into the adjacent domain of infrastructure management. The Network Management service makes VLAN provisioning and bridge configuration accessible without requiring root-level CLI access, while the Remote Desktop service eliminates the context-switching overhead of maintaining separate SSH and RDP clients. The cross-service synergies observed during evaluation particularly between the Network Manager and Docker Compose Builder, and between the System Monitor and Remote Desktop shortcut confirm that integration yields qualitative usability benefits that are not achievable by standalone tools.

REFERENCES

1. D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," Linux Journal, vol. 2014, no. 239, Mar. 2014.
2. D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," IEEE Cloud Computing, vol. 1, no. 3, pp. 81–84, Sep. 2014.
3. Python Software Foundation, "venv Creation of virtual environments," Python 3 Documentation, 2024. [Online]. Available: <https://docs.python.org/3/library/venv.html>
4. asdf Contributors, "asdf Manage multiple runtime versions with a single CLI tool," 2023. [Online]. Available: <https://asdf-vm.com>
5. N. D. Matsakis and F. S. Klock II, "The Rust Language," in Proc. ACM HILT, 2014, pp. 103–104.



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

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

6. Tokio Contributors, "Tokio: An asynchronous runtime for the Rust programming language," 2024. [Online]. Available: <https://tokio.rs>
7. GitHub Inc., "Electron: Build cross-platform desktop apps with JavaScript, HTML, and CSS," 2024. [Online]. Available: <https://www.electronjs.org>
8. Tauri Contributors, "Tauri: Build smaller, faster, and more secure desktop applications with a web frontend," 2024. [Online]. Available: <https://tauri.app>
9. Dioxus Contributors, "Dioxus: A portable, performant, and ergonomic framework for building user interfaces in Rust," 2024. [Online]. Available: <https://dioxuslabs.com>
10. iproute2 Contributors, "iproute2: Utilities for Linux networking," 2024. [Online]. Available: <https://wiki.linuxfoundation.org/networking/iproute2>
11. T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," RFC 4251, Internet Engineering Task Force, Jan. 2006.
12. Microsoft Corporation, "Remote Desktop Protocol (RDP) Overview," Windows Server Documentation, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/windows-server/remote/remote-desktop-services/>



INTERNATIONAL
STANDARD
SERIAL
NUMBER
INDIA



SJIF Scientific Journal Impact Factor



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