Archives available at journals.mriindia.com

International Journal on Advanced Computer Theory and Engineering

ISSN: 2319-2526

Volume 15 Issue 01, 2026

Real-Time Docker Container Monitoring System Using Terminal-Based User Interface

[1]Umang Patel, [2]Harsha Dubey
[1,2]Department of CSE, SSIPMT, Raipur
[1]umang@ssipmt.com, [2]harsha.d@ssipmt.com

Peer Review Information

Submission: 29 March 2026
Revision: 17 April 2026
Acceptance: 21 May 2026

Keywords

Containerization, Docker Monitoring, Resource Visualization, System administration

Abstract

The huge adaptation for containerization is a very huge leap in computer science and computer technologies and has a huge need for optimized dashboards for major technological stacks. This project aims for providing a robust analytical framework and architecture for the projects that need this should work flawlessly as the development team could just display the data on a single page on their main server display for constant monitoring and testing for their development builds this is very crucial for people who are working with clusters who have managed, restart or check network data for data transfers on the wire

Introduction

Docker with its base concept of containerization has changed and transformed how applications are monitored and managed through various isolation and containerization [1], [2]. But cluster management of these lightweight containers are necessary for finding the bug and managing individual docker images.

Various monitoring tools today are developed for these technologies that provide a robust framework for managing unique components [5], [6]. My implementation takes that further by allowing it to run on any light or heavy machine which is used for deploying the containers. It is a very lightweight python script which uses the docker python api library [11] for managing containers that have been deployed on the machine. This works as a dashboard which can be used through ssh and then used to monitor the complete machine configuration, container specific details such as processor usage percentage and network details.

This implementation can be used to trigger start and stop commands for the container which is important if someone needs to restart the container at the point of encountering the bug.

This project developed a real-time monitoring system that collects Docker data through the terminal interface. Our goal was to give developers immediate visibility into container performance without requiring additional infrastructure or graphical environments

Literature Review

Docker was released in 2013 [2] and hence the container monitoring has leaped considerably. Early work examined monitoring through os-level virtualization [3]. Currently the monitoring systems use the linux kernel cgroup mechanism to track everything [4].

In python3 we can implement these metrics through its stats api endpoint [11], providing data on cpu usage, memory consumption, network interface and block i/o operations. This api lets us use and collect these metrics using very minimal system resources.

This project utilizes the textual framework [9], [10] which is a python library for creating and working with tui libraries. The reason for choosing tui over web-based gui or native gui is because of speed, size and performance. As terminal as lightweight and can be accessed via ssh they are great for monitoring the server

system.
Real time data visualization is a major paper of this project and has the management and conditioning for this container as not completely preconfigured things go wrong and then the whole architecture is broken hence the real time monitoring and also necessary for taking quick steps on the spot [7], [8].

Methodology
Our monitoring system uses a three-tier architecture: data collection, processing, and presentation. The collection layer uses the docker api which is provided in the docker sdk [11] for developing the collection layer for the system – wide docker info using these endpoints we collect and present the data.
This project connects to the docker daemon using docker.from_env() which detects sockets locations automatically through the operating system [11]. This collects the containers through container.list(all=True) getting all the running and stopped containers. The DataTable widget [10] shows container information: short id, name, image source and status. Through the textual event system we capture DataTable.RowSelected events to identify the selection and store the container id for further actions and operations.To avoid blocking the main UI thread, we use a worker thread for continuous metrics collection. The @work(exclusive=True) decorator [10] ensures only one monitoring worker runs at a time, cancelling previous workers when users change container selections. The collection for the cpu usage is done using docker's algorithm [11] for calculating ratio of container cpu time to system cpu time:
cpu_percent = (cpu_delta / system_delta) × cpu_count × 100
Here, cpu_delta is the container's CPU usage change, system_delta is the system-wide CPU time change, and cpu_count is the number of available processor cores.
This project is implemented using two main display mechanisms [10]. A Sparkline widget shows CPU utilization history, keeping sixty data points to provide one minute of context at two-second intervals. We use the sparkline max summary function for getting the peak reading for the usage of resources.
Memory consumption is displayed using ProgressBar widget [10] with total range to container memory limit and current progress shows actual usage, the label displays the conversion from bytes to megabytes.
The logging part of this project is very important as the log for containers work like black box for this container and can point out the bugs in the system. The project is focused mostly for monitoring and hence the python script can export logs into a text format which can be used later and also used for sharing as a report. The logs are syntax-highlight and when checked presents a great overview for the container.
This script follows and tries to emulate the unix keyboard navigation conventions for faster usage. These keyboard hotkeys are made for faster access and memorisation for everyday monitoring use. Key bindings include 'q' to quit, 'r' to refresh the container list, 's' to start selected containers, and 'x' to stop them. These bindings appear in the footer for easy reference.

Results
Our monitoring system uses a three-tier architecture: data collection, processing, and presentation. The collection layer uses the docker api which is provided in the docker sdk [11] for developing the collection layer for the system – wide docker info using these endpoints we collect and present the data.
This project tested the system with docker [2]. The application successfully identified containers in multiple states: running, paused, and exited. The project data calculation is done in a format that matches the docker style of information management [11].
The script is stable even on long sustained usage the cpu and memory usage for the project remains stable and efficient providing a very minimal overhead for the system [5] running at a stable 35 megabytes for the whole duration this is due to the simple architecture and stability for the tui [10]. Using this during an ssh session for overview works perfectly for checking the stats and checking the logs for orchestrating the containers and the clusters.
The sparkline tui elements are great for accessing and glancing the data for the containers the script shows [7], [8]. The project is running at a very stable frame per second and this speed can be used to manage multiple container instances.
The system fits many real development and maintenance systems [6]. In the devops pipeline, it can help developers during the CI/CD deployments, helping identify misconfiguration services or resource bottlenecks early in the integration process. In production debugging scenarios the ability to instantly access real time resources analytics.

Conclusion
The project successfully developed a terminal based docker management system with proper emphasis on the monitoring of docker containers for development and deployment [1],

[2]. The project meets the requirement for the robust monitoring system for host machines. The project does continuous metrics collection, responsive visualization, and interactive container control. The project was aimed to provide data over ssh for remote container management and also implemented as a dashboard for checking logs and activity during the execution for the docker container management. The project is made as modular as possible and hence can be improved further upon for more features the script is made standalone for the use minimal deployment and setup time this is very necessary considering the high speed working and data management for multiple clusters for docker. On general purpose this is a dashboard which can be configured to focus on the issue that might be happening during the testing or deployment of the application.

The major impact and goal for this research is through learning and development for Docker. This project was aimed to provide a fast, minimal, performant and intuitive approach for learning and development [9], [10]. The project while on current performance has accomplished this with a very low resource overhead [5]. While several docker monitoring tools already exist [6], they need prior setup and additional overhead for the program.

References

D. Bernstein, "Containers and cloud: From LXC to Docker to Kubernetes," IEEE Cloud Computing, vol. 1, no. 3, pp. 81–84, 2014.

D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," Linux Journal, vol. 2014, no. 239, p. 2, 2014.

S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors," ACM SIGOPS Operating Systems Review, vol. 41, no. 3, pp. 275–287, 2007.

P. B. Menage, "Adding generic process containers to the Linux kernel," in Proc. Linux Symposium, vol. 2, 2007, pp. 45–57.

E. Casalicchio and V. Perciballi, "Measuring Docker performance: What a mess!" in Proc. ACM/SPEC Int. Conf. Performance Engineering Companion (ICPE '17 Companion), 2017, pp. 11–16.

C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: A state-of-the-art review," IEEE Trans. Cloud Comput., vol. 7, no. 3, pp. 677–692, Sep./Oct. 2019.

J. Heer and M. Bostock, "Crowdsourcing graphical perception: Using Mechanical Turk to assess visualization design," in Proc. SIGCHI Conf. Human Factors in Comput. Syst. (CHI '10), 2010, pp. 203–212.

E. R. Tufte, The Visual Display of Quantitative Information, 2nd ed. Cheshire, CT, USA: Graphics Press, 2001.

S. Willison, "Building rich terminal dashboards with Python and Textual," in Proc. PyCon US, 2022, p. 156.

Textualize, "Textual: Rapid application development framework for Python," 2022. [Online]. Available: https://textual.textualize.io/

Docker Inc., "Docker Engine API documentation," 2020. [Online]. Available: https://docs.docker.com/engine/api/