Deep Dive : A System Resource Monitor

Operating Systems (IT253) Course Project

Submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY

by

NITHIN S	221IT085
AYUSH KUMAR	221IT015
JAY CHAVAN	221IT020

DEPARTMENT OF INFORMATION TECHNOLOGY

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL, MANGALORE -575025

March, 2024

D E C L A R A T I O N

We hereby *declare* that the *OS Project Report* entitled **“Deep Dive : A System Resource Monitor”** which is being submitted to the National Institute of Technology Karnataka Surathkal, in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in the Department of Information Technology, is a ***bonafide report of the work carried out by us.*** The material contained in this project report has not been submitted to any University or Institution for the award of any degree.

Nithin S	Ayush Kumar	Jay Chavan
Signature Department of IT	Signature Department of IT	Signature Department of IT

Place	:	NITK, SURATHKAL
Date	:	25 March 2024

CERTIFICATE

This is to certify that the Seminar entitled **"Deep Dive : A System Resource Monitor"** has been presented by Nithin S (221IT085), Ayush Kumar (221IT015) & Jay Chavan (221IT020), students of IV semester B.Tech.(I.T), Department of Information Technology, National Institute of Technology Karnataka, Surathkal, on 25 March, 2024, during the even semester of the academic year 2023 - 2024, in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Information Technology.

Examiner-1 Name	Examiner-2 Name
Signature of the Examiner-1 with Date	Signature of the Examiner-2 with Date

Place:

Date:

ABSTRACT

This report presents Deep Dive, a System Resource Monitor which is a sophisticated software project aimed at providing users with an intuitive and comprehensive tool for monitoring system resources in real-time. Leveraging the power of Python and Qt GUI interface, this project offers a user-friendly experience while ensuring accurate tracking and analysis of critical system metrics. The system resource monitor provides insights into CPU usage, memory consumption, disk activity, network traffic, and other vital parameters, enabling users to identify performance bottlenecks, optimize resource utilization, and troubleshoot potential issues efficiently. With its responsive and visually appealing interface, Deep Dive empowers both novice and advanced users to make informed decisions about system resource management, thereby enhancing system stability, reliability, and overall performance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

In the dynamic realm of modern computing, the efficient management of system resources emerges as a critical necessity, underpinning the smooth operation and longevity of computing environments across diverse domains. As the complexity of software applications escalates and the demands on hardware intensify, the need for sophisticated tools that facilitate real-time monitoring and analysis of system resources becomes increasingly pronounced. It is within this context that our project, "Deep Dive: A System Resource Monitor," assumes significance. Deep Dive represents a concerted effort to address the multifaceted challenges associated with system resource management by providing users with a comprehensive and intuitive platform for monitoring and analyzing crucial system metrics. By harnessing the capabilities of Python for robust backend functionality and Qt GUI interface for seamless user interaction, Deep Dive endeavors to redefine the paradigm of system resource monitoring, offering users unprecedented insights into CPU usage, memory consumption, disk activity, network traffic, and other key performance indicators.

Language Used : Python

Libraries Used : Psutil, Os, PyQt5, PyCPUInfo, Distro, Numpy

Working Environment : Any Linux Distro

Requirements : A Linux Environment, Numpy, Psutil, Distro, PyQT5, PyQtGraph, PyCPUinfo, PyQTChart

CHAPTER 2: OBJECTIVE

At the core of Deep Dive lies a steadfast commitment to empowering users with the tools and insights necessary to optimize system resources effectively. Our project sets out with a singular objective: to equip users with the means to not merely monitor but also proactively manage system resources in real-time. Through the provision of actionable insights into critical performance metrics, Deep Dive enables users to identify bottlenecks, fine-tune resource allocation, and swiftly address any arising performance issues. By fostering a deeper understanding of system resource utilization patterns and trends, the tool aims to enhance system efficiency, reliability, and overall performance. Through its intuitive interface and advanced visualization techniques, Deep Dive seeks to democratize system resource management, making it accessible and comprehensible to users across diverse technical backgrounds and proficiency levels.

The objective is to explore all these fields:

- **CPU:** Percentage, Clock Speeds, Occupancy, Duration, Counts of Context Switches, System calls and hardware interrupts
- **Memory:** Utilization, Swapping rate, Occupancy, Physical and logical partition.
- **Internet:** Packet Count, Size, Rate, Transfer rate, Transmission, Dropped transfers.
- **Processes:** PIDs, names, terminal, usernames, states, CPU/Memory Usage, context switches, threads, Kill, resume, Terminate, Suspend Options.
- **System/Hardware:** OS, Kernel names, Versions, CPU name, Vendor, Frequency, Features.
- Visually represent real time data changes for clear understanding

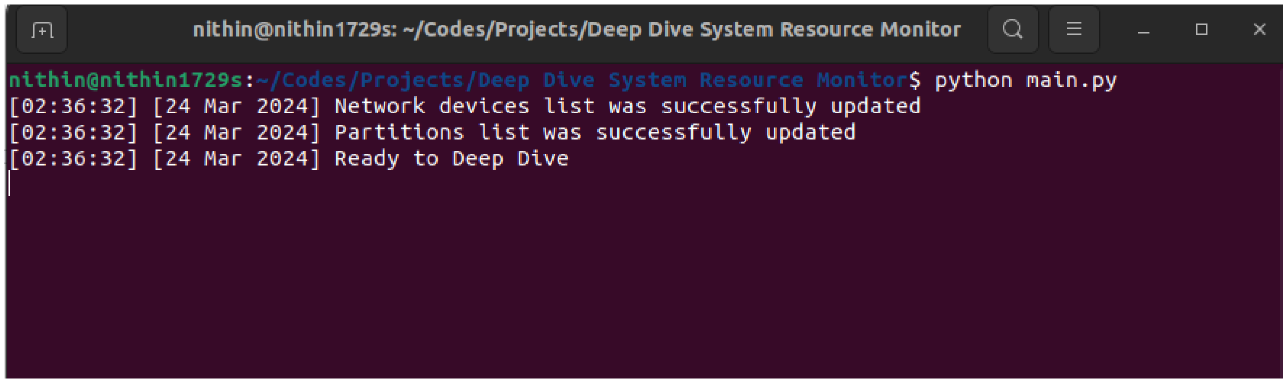
The main goal is to integrate all these metrics into one application and give a naive user all the in depth details about his/her system, which is not done by conventional system monitors.

CHAPTER 3: METHODOLOGY

The development of Deep Dive is underpinned by a meticulously crafted methodology that combines rigorous data collection, sophisticated analysis, and intuitive presentation of insights. Leveraging Python's versatility and robustness, the tool employs a multifaceted approach to gather, process, and analyze data pertaining to various system metrics, including CPU usage, memory utilization, disk activity, network statistics, and process details. Through the adept utilization of algorithms and techniques, Deep Dive ensures the accuracy, responsiveness, and reliability of resource monitoring, even in the face of dynamic and rapidly changing computing environments. Real-time visualization of this data constitutes a cornerstone of the tool, facilitating instantaneous insights into system performance and enabling users to make informed decisions regarding resource management. By adhering to a structured methodology that prioritizes user-centric design principles and robust backend functionality, Deep Dive endeavors to deliver a seamless and immersive experience that empowers users to unlock the full potential of their computing environments.

Code Structure

1. **commons:** Consist for python code for the frontend of the application. Its designed using Qt5 Designer
2. **readers:**
    - **mainwind** : Stands for the main window. The data for all 6 tabs are collected here.
    - **procwind** : Stands for the process window. It provides user interaction for process control.
3. **screens:**
    - **mainwind** : Consists of interface design for all the tabs.
    - **procwind** : Consists of interface design for the mini process tabs.
4. **widgets** :
    - **lgptwdgt, ntwkwdgt, perfwdgt, phptwdgt** : All these are used to design widgets for tabs


nithin@nithin1729s: ~/Codes/Projects/Deep Dive System Resource Monitor

```
nithin@nithin1729s:~/Codes/Projects/Deep Dive System Resource Monitor$ python main.py
[02:36:32] [24 Mar 2024] Network devices list was successfully updated
[02:36:32] [24 Mar 2024] Partitions list was successfully updated
[02:36:32] [24 Mar 2024] Ready to Deep Dive
```

Figure 3.1: Terminal Activity of the application

Deep Dive employs a variety of techniques for data collection to ensure comprehensive coverage of system metrics. This includes interfacing with operating system APIs to retrieve real-time data on CPU usage, memory utilization, disk activity, network statistics, and process details. Additionally, the tool may utilize performance monitoring tools or libraries to access low-level system information with precision and accuracy.

Deep Dive prioritizes real-time visualization as a key component of its user interface. Utilizing modern visualization libraries and techniques, the tool presents system metrics in a visually appealing and intuitive manner. This allows users to quickly grasp the current state of their system's performance and identify any areas requiring attention. Interactive elements may be incorporated to enable users to drill down into specific metrics or timeframes for deeper analysis. Deep Dive is designed to adapt seamlessly to dynamic and rapidly changing computing environments. This adaptability is achieved through robust error handling, graceful degradation, and automatic recovery mechanisms to maintain uninterrupted monitoring even in the face of transient errors or system fluctuations.

CHAPTER 4: SYSTEM DESIGN

Block Diagram

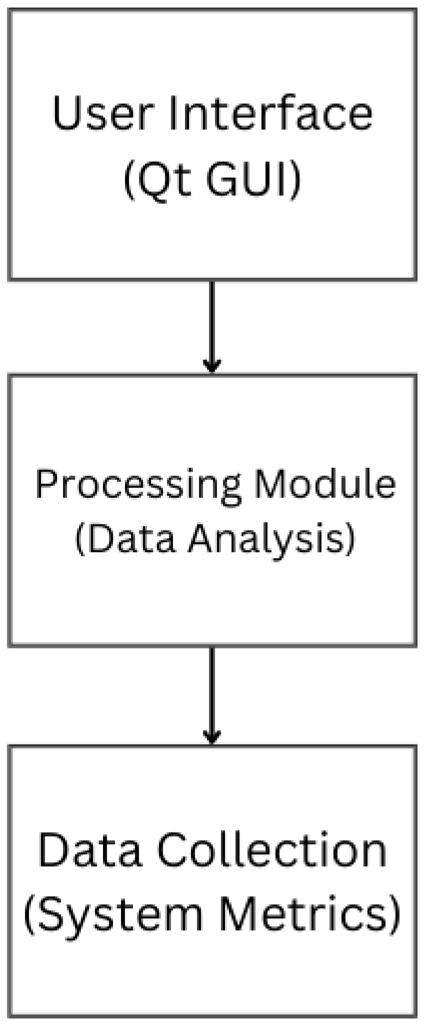


Figure 4.1: Flowchart of Application Architecture

The Data Collection is done by Python libraries like Psutil, PyCPUInfo, Distro and os. These raw system metrics are analyzed to prepare meaningful statistics in the processing module. More information regarding this is given in the next chapter.

These processed stats are displayed in a user-friendly manner by the PyQt interface.

CHAPTER 5: IMPLEMENTATION

In this chapter, we will discuss how the various statistics are collected by our System Resource Monitor using Python Libraries.

Description	Command Used
Name of the Linux Distro	Distro.name()
Version of the Linux Distro	distro.version()
Hostname of the system	uname().nodename
Release of the OS	uname().version
Version information of the operating system	uname().version
Boot time of the system	psutil.boot_time()
Bytes received	nxntiocf.bytes_recv - pvntiocf.bytes_recv
Bytes sent	nxntiocf.bytes_sent - pvntiocf.bytes_sent
Packets received	nxntiocf.packets_recv - pvntiocf.packets_recv
Packets sent	nxntiocf.packets_sent - pvntiocf.packets_sent
Bytes received per network interface	nxntioct[indx].bytes_recv - pvntioct[indx].bytes_recv
Bytes sent per network interface	nxntioct[indx].bytes_sent - pvntioct[indx].bytes_sent
Packets received per network interface	nxntioct[indx].packets_recv - pvntioct[indx].packets_recv
Packets sent per network interface	nxntioct[indx].packets_sent - pvntioct[indx].packets_sent
Total bytes received	netiocnf.bytes_recv
Total bytes sent	netiocnf.bytes_sent
Total packets received	netiocnf.packets_recv
Total packets sent	netiocnf.packets_sent

Receive errors	netiocnf.errin
Send errors	netiocnf.errout
Dropped packets on receive	netiocnf.dropin
Dropped packets on send	netiocnf.dropout
Disk read count	diskioqt.read_count
Disk write count	diskioqt.write_count
Bytes read from disk	mmrysize.format(diskioqt.read_bytes)
Bytes written to disk	mmrysize.format(diskioqt.write_bytes)
Time spent reading from disk	timevalu.format(diskioqt.read_time)
Time spent writing to disk	timevalu.format(diskioqt.write_time)
Merged reads	diskioqt.read_merged_count
Merged writes	diskioqt.write_merged_count
Number of disk partitions	len(partlist)
Logical partition device	indx.device
Logical partition usage	{ "free": mmrysize.format(partfree), "used": mmrysize.format(partused), "comp": mmrysize.format(partcomp), "perc": partperc }
Logical partition filesystem information	{ "mtpt": indx.mountpoint, "fsys": indx.fstype }
Physical partition device	indx.device
Physical partition usage	{ "free": mmrysize.format(partfree), "used": mmrysize.format(partused), "comp": mmrysize.format(partcomp), "perc": partperc }
Physical partition filesystem information	{ "mtpt": indx.mountpoint, "fsys": indx.fstype }
CPU utilization percentage	int(cpudperc[indx])
Maximum CPU frequency	freqvalu.format(cpudfreq[indx].max)
Minimum CPU frequency	freqvalu.format(cpudfreq[indx].min)

Current CPU frequency	freqvalu.format(cpudfreq[indx].current)
Time spent in user mode	timevalu.format(cputsecs[indx].user)
Time spent in user mode with low priority	timevalu.format(cputsecs[indx].nice)
Time spent in kernel mode	timevalu.format(cputsecs[indx].system)
Time spent idle	timevalu.format(cputsecs[indx].idle)
Time spent in I/O wait	timevalu.format(cputsecs[indx].iowait)
Time spent handling hardware interrupts	timevalu.format(cputsecs[indx].irq)
Time spent handling software interrupts	timevalu.format(cputsecs[indx].softirq)
Time spent by other operating systems running in a virtualized environment	timevalu.format(cputsecs[indx].steal)
Time spent running a guest operating system	timevalu.format(cputsecs[indx].guest)
Time spent running a niced guest	timevalu.format(cputsecs[indx].guest_nice)
User CPU time percentage	cputperc[indx].user
User CPU time percentage with low priority	cputperc[indx].nice
System CPU time percentage	cputperc[indx].system
Idle CPU time percentage	cputperc[indx].idle
I/O wait CPU time percentage	cputperc[indx].iowait
Hardware interrupt CPU time percentage	cputperc[indx].irq
Software interrupt CPU time percentage	cputperc[indx].softirq
Steal CPU time percentage	cputperc[indx].steal
Guest CPU time percentage	cputperc[indx].guest
Niced guest CPU time percentage	cputperc[indx].guest_nice
CPU name	get_cpu_info().get("brand_raw")
CPU vendor ID	get_cpu_info().get("vendor_id_raw")
CPU frequency	get_cpu_info().get("hz_advertised_friendly")
Number of CPU cores	get_cpu_info().get("count")

CPU architecture (32-bit/64-bit)	get_cpu_info().get("bits")
CPU architecture	get_cpu_info().get("arch")
CPU stepping	get_cpu_info().get("stepping")
CPU model	get_cpu_info().get("model")
CPU family	get_cpu_info().get("family")
CPU feature flags	get_cpu_info().get("flags")
Process ID	str(proc.info["pid"])
Process name	str(proc.info["name"])
Terminal associated with the process	str(proc.info["terminal"])
User running the process	str(proc.info["username"])
Process status (running, sleeping, etc.)	str(proc.info["status"])
CPU usage percentage by the process	%2.1f" % proc.info["cpu_percent"]
Memory usage percentage by the process	"%2.1f" % proc.info["memory_percent"]
Number of threads created by the process	str(proc.info["num_threads"])
Username of the current user	getpass.getuser()
CPU usage percentage	psutil.cpu_percent()
Memory usage percentage	psutil.virtual_memory().percent
Swap memory usage percentage	psutil.swap_memory().percent
Disk usage percentage of the root directory	psutil.disk_usage("/").percent
Percentage of used memory	psutil.virtual_memory().used * 100 / psutil.virtual_memory().total
Percentage of cached memory	psutil.virtual_memory().cached * 100 / psutil.virtual_memory().total
Percentage of free memory	psutil.virtual_memory().free * 100 / psutil.virtual_memory().total
Absolute value of used memory	mmrysize.format(psutil.virtual_memory().used)
Absolute value of cached memory	mmrysize.format(psutil.virtual_memory().cache)

Absolute value of free memory	mmrysize.format(psutil.virtual_memory().free)
Absolute value of total memory	mmrysize.format(psutil.virtual_memory().total)
Absolute value of active memory	mmrysize.format(psutil.virtual_memory().acti ve)
Absolute value of memory buffers	mmrysize.format(psutil.virtual_memory().buff ers)
Absolute value of shared memory	mmrysize.format(psutil.virtual_memory().shar ed)
Absolute value of slab memory	mmrysize.format(psutil.virtual_memory().slab)
Percentage of used swap memory	(psutil.swap_memory().used * 100 / psutil.swap_memory().total) if psutil.swap_memory().total > 0 else 0
Percentage of free swap memory	(psutil.swap_memory().free * 100 / psutil.swap_memory().total) if psutil.swap_memory().total > 0 else 100
Absolute value of used swap memory	mmrysize.format(psutil.swap_memory().used)
Absolute value of free swap memory	mmrysize.format(psutil.swap_memory().free)
Absolute value of total swap memory	mmrysize.format(psutil.swap_memory().total)
Absolute value of swap memory in	mmrysize.format(psutil.swap_memory().sin)
Absolute value of swap memory out	mmrysize.format(psutil.swap_memory().sout)
Percentage of CPU usage	psutil.cpu_percent()
Percentage of free CPU usage	100 - psutil.cpu_percent()
Number of context switches since boot	psutil.cpu_stats().ctx_switches
Number of interrupts serviced since boot	psutil.cpu_stats().interrupts
Number of software interrupts since boot	psutil.cpu_stats().soft_interrupts
Number of system calls since boot	psutil.cpu_stats().syscalls
Process name	procobjc.name()

CPU usage percentage by the process	procobjc.cpu_percent()
Memory usage percentage by the process	procobjc.memory_percent()
CPU number where the process is running	procobjc.cpu_num()
Number of threads created by the process	procobjc.num_threads()
User running the process	procobjc.username()
Terminal associated with the process	procobjc.terminal()
Niceness value of the process	procobjc.nice()
I/O scheduling class of the process	procobjc.ionice().value
Number of context switches experienced by the process	procobjc.num_ctx_switches().voluntary, procobjc.num_ctx_switches().involuntary
Parent process ID	procobjc.ppid()
Process status (running, sleeping, etc.)	procobjc.status().title(
Time when the process was created	procobjc.create_time()
Time when the statistics were acquired	time.time()

Table 5.1: Origin of Statistics

Many more metrics are displayed in the application which has a similar origin. Since a lot of function calls are done repeatedly to acquire these metrics the application gets overhead and a high startup time.

This might be improved by implementing multithreading and also making some changes in the architecture of our application, which is a future scope.

These docs below are referred extensively to prepare the resource monitor application.

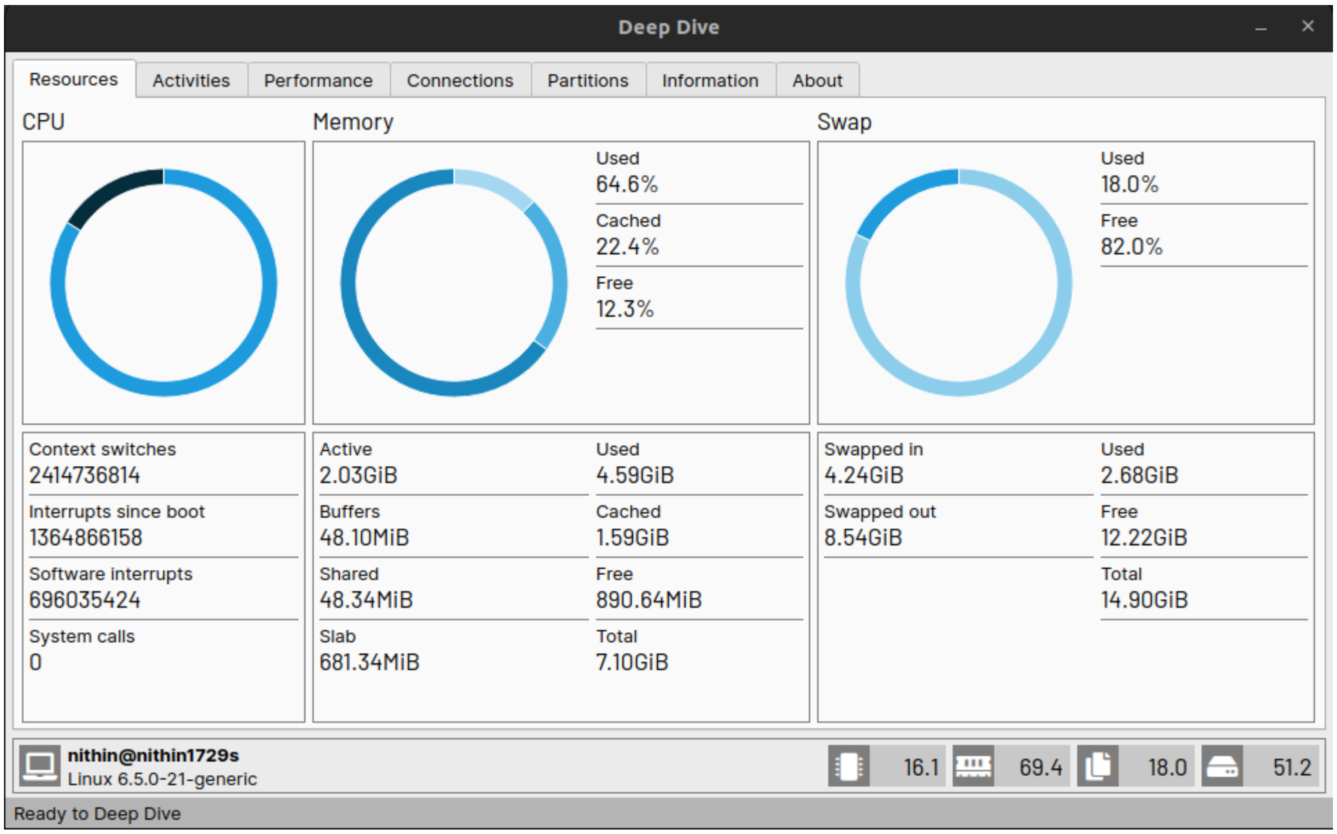
PSutil : https://psutil.readthedocs.io/en/latest/
PyCPUInfo : https://github.com/workhorsy/py-cpuinfo
os: https://docs.python.org/3/library/os.html
Distro : https://distro.readthedocs.io/en/latest/

CHAPTER 6: CONCLUSION

RESULTS

Resource Tabscreen :

- Displays CPU, Memory, and Swap Memory usage using a radial pie chart.
- Bottom status bar shows CPU Percentage, Memory Percentage, Swap Percentage, and Disk Percentage, along with Linux Kernel Version and owner's name.
- The number of system calls is always set to 0 in Linux.



CPU

Memory

Used
64.6%

Cached
22.4%

Free
12.3%

Swap

Used
18.0%

Free
82.0%

Context switches
2414736814

Interrupts since boot
1364866158

Software interrupts
696035424

System calls
0

Active
2.03GiB

Buffers
48.10MiB

Shared
48.34MiB

Slab
681.34MiB

Used
4.59GiB

Cached
1.59GiB

Free
890.64MiB

Total
7.10GiB

Swapped in
4.24GiB

Swapped out
8.54GiB

Used
2.68GiB

Free
12.22GiB

Total
14.90GiB

nithin@nithin1729s
Linux 6.5.0-21-generic

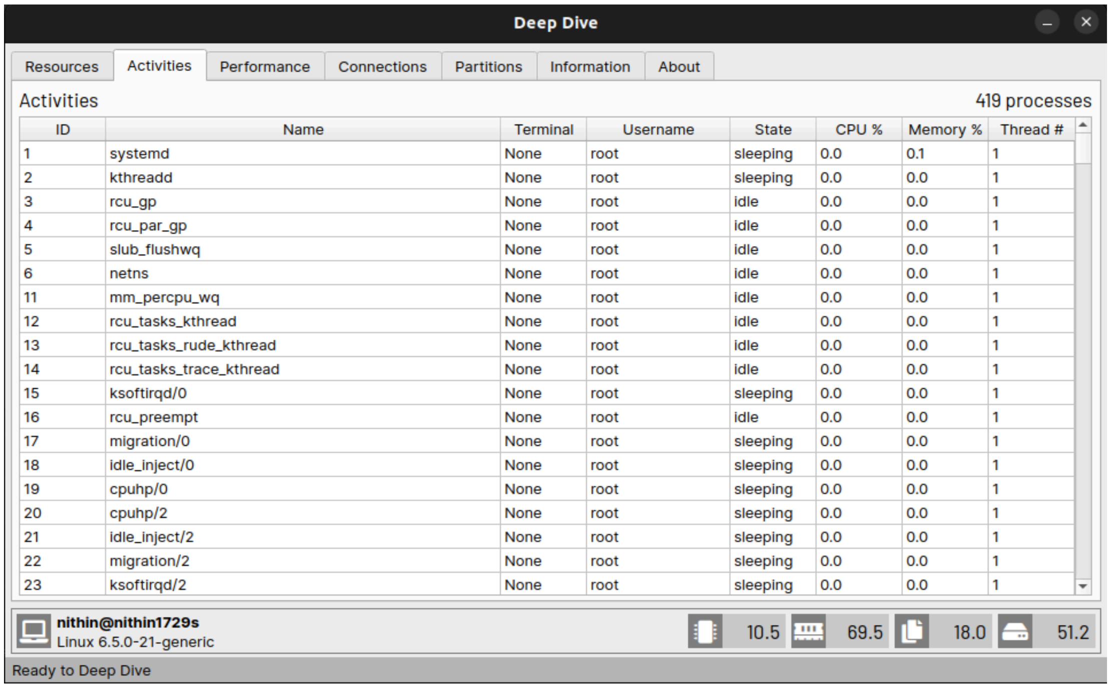
16.1 69.4 18.0 51.2

Ready to Deep Dive

Figure 6.1: Resources Tab Screen

Activities Tabscreen :

- Lists processes currently running on the local machine.
- Provides control over terminating, suspending, resuming processes.
- Shows details about the user who created the process, Process ID, Process's Parent ID, CPU % used by the process, etc.

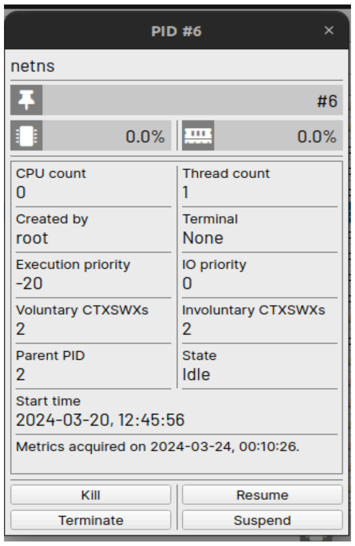


Activities 419 processes

ID	Name	Terminal	Username	State	CPU %	Memory %	Thread #
1	systemd	None	root	sleeping	0.0	0.1	1
2	kthreadd	None	root	sleeping	0.0	0.0	1
3	rcu_gp	None	root	idle	0.0	0.0	1
4	rcu_par_gp	None	root	idle	0.0	0.0	1
5	slub_flushwq	None	root	idle	0.0	0.0	1
6	netns	None	root	idle	0.0	0.0	1
11	mm_percpu_wq	None	root	idle	0.0	0.0	1
12	rcu_tasks_kthread	None	root	idle	0.0	0.0	1
13	rcu_tasks_rude_kthread	None	root	idle	0.0	0.0	1
14	rcu_tasks_trace_kthread	None	root	idle	0.0	0.0	1
15	ksoftirqd/0	None	root	sleeping	0.0	0.0	1
16	rcu_preempt	None	root	idle	0.0	0.0	1
17	migration/0	None	root	sleeping	0.0	0.0	1
18	idle_inject/0	None	root	sleeping	0.0	0.0	1
19	cpuhp/0	None	root	sleeping	0.0	0.0	1
20	cpuhp/2	None	root	sleeping	0.0	0.0	1
21	idle_inject/2	None	root	sleeping	0.0	0.0	1
22	migration/2	None	root	sleeping	0.0	0.0	1
23	ksoftirqd/2	None	root	sleeping	0.0	0.0	1

nithin@nithin1729s
Linux 6.5.0-21-generic
10.5 69.5 18.0 51.2
Ready to Deep Dive

Figure 6.2: Activities Tab Screen



netns
#6
0.0% 0.0%

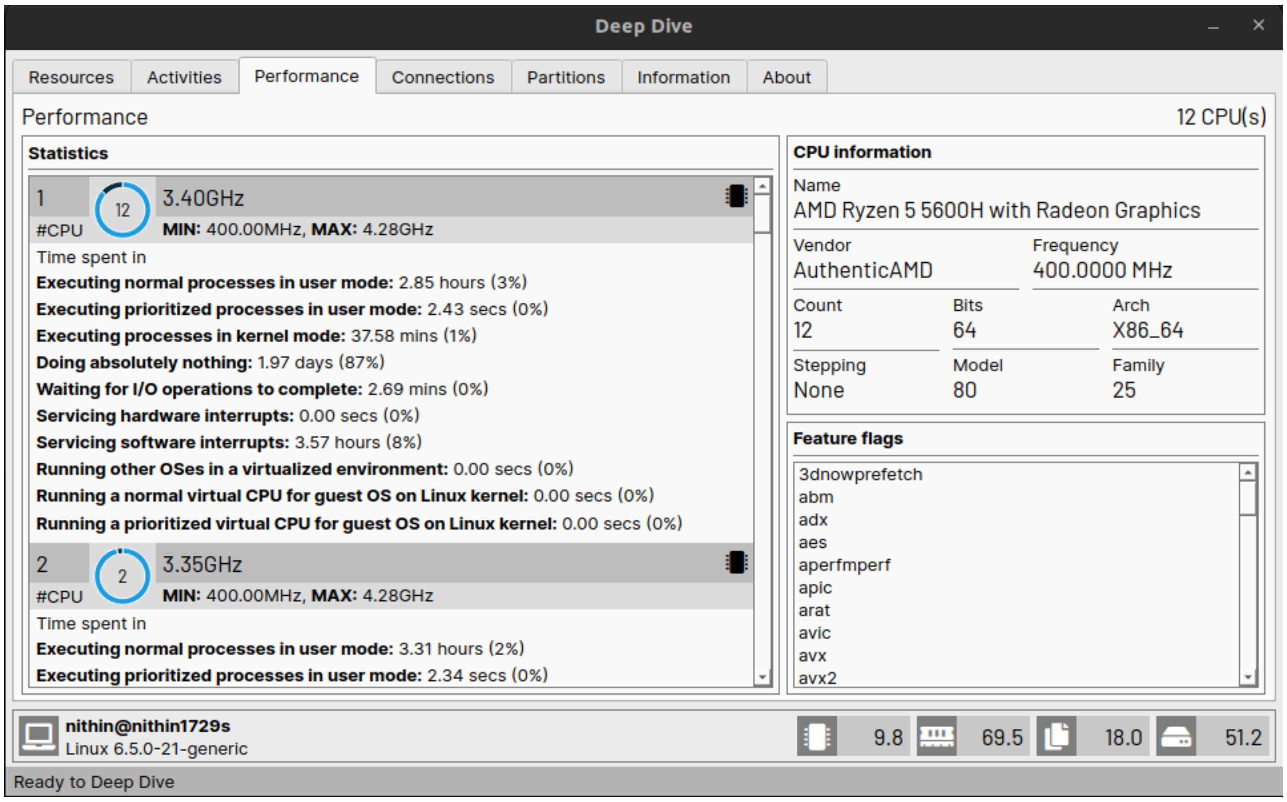
CPU count 0	Thread count 1
Created by root	Terminal None
Execution priority -20	IO priority 0
Voluntary CTXSWXs 2	Involuntary CTXSWXs 2
Parent PID 2	State Idle
Start time 2024-03-20, 12:45:56	

Metrics acquired on 2024-03-24, 00:10:26.

Figure 6.3: Process Mini tab Screen

Performance Tabscreen :

- Provides information regarding various statistics for each core of the processor.
- Displays CPU Name, Vendor, Count, Architecture, Feature flags, etc.
- Feature flags represent specific capabilities or features supported by the CPU.



Performance 12 CPU(s)

Statistics

1 #CPU 12 3.40GHz
MIN: 400.00MHz, **MAX:** 4.28GHz
Time spent in
Executing normal processes in user mode: 2.85 hours (3%)
Executing prioritized processes in user mode: 2.43 secs (0%)
Executing processes in kernel mode: 37.58 mins (1%)
Doing absolutely nothing: 1.97 days (87%)
Waiting for I/O operations to complete: 2.69 mins (0%)
Servicing hardware interrupts: 0.00 secs (0%)
Servicing software interrupts: 3.57 hours (8%)
Running other OSes in a virtualized environment: 0.00 secs (0%)
Running a normal virtual CPU for guest OS on Linux kernel: 0.00 secs (0%)
Running a prioritized virtual CPU for guest OS on Linux kernel: 0.00 secs (0%)

2 #CPU 2 3.35GHz
MIN: 400.00MHz, **MAX:** 4.28GHz
Time spent in
Executing normal processes in user mode: 3.31 hours (2%)
Executing prioritized processes in user mode: 2.34 secs (0%)

CPU information

Name
AMD Ryzen 5 5600H with Radeon Graphics

Vendor	Frequency	
AuthenticAMD	400.0000 MHz	
Count	**Bits**	**Arch**
12	64	X86_64
Stepping	**Model**	**Family**
None	80	25

Feature flags

3dnowprefetch
abm
adx
aes
aperfmperf
apic
arat
avic
avx
avx2

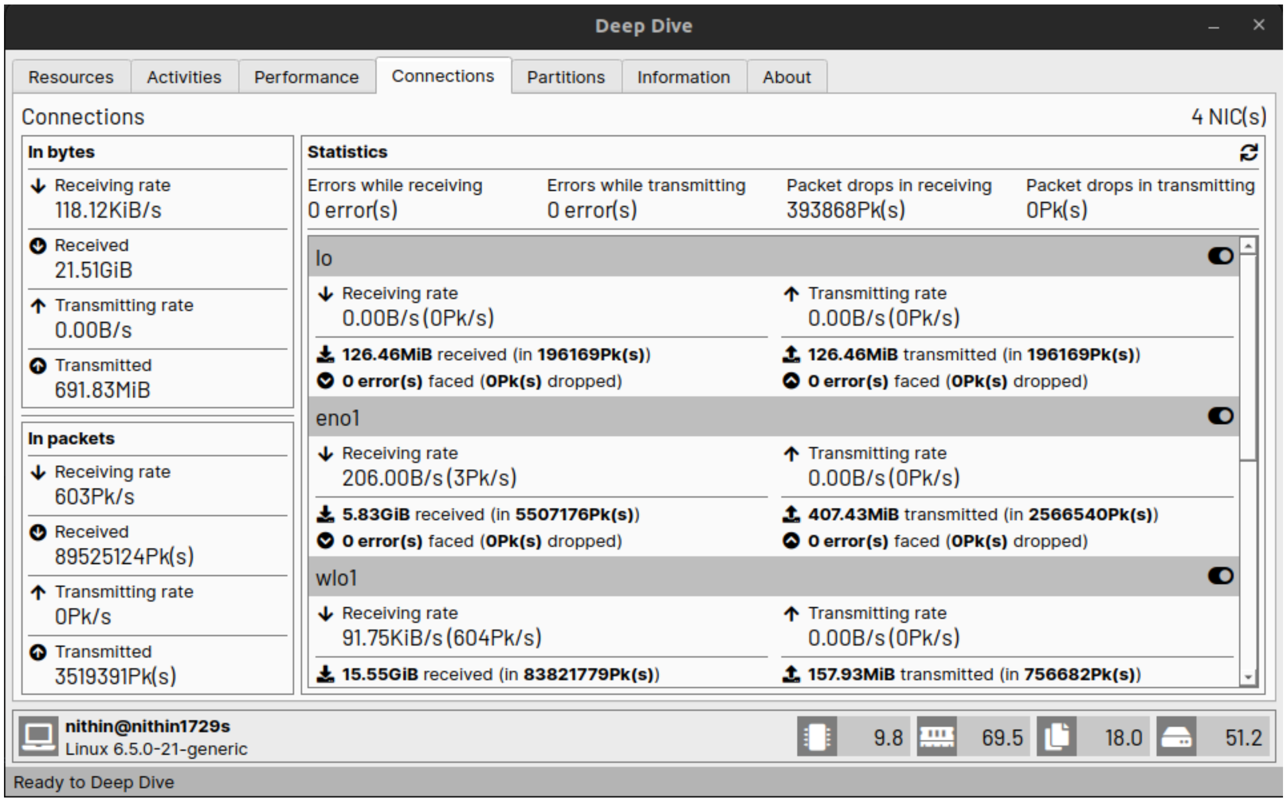
nithin@nithin1729s
Linux 6.5.0-21-generic
9.8 69.5 18.0 51.2
Ready to Deep Dive

Figure 6.4: Performance Tab Screen

Feature flags, also known as CPU flags or CPU feature flags, are indicators that represent specific capabilities or features supported by a CPU (Central Processing Unit). These flags are used by operating systems, software, and sometimes even by other hardware components to determine the available features and optimize performance accordingly.

Connection Tab Screen :

- Displays statistics about internet speed and keeps track of the number of bytes or packets received, and data uploaded or downloaded.
- Includes information about all Network Interface Cards (NIC) present in the system.
- Provides a refresh button to update and view new internet connections.



Connections 4 NIC(s)

In bytes

Receiving rate
118.12KiB/s

Received
21.51GiB

Transmitting rate
0.00B/s

Transmitted
691.83MiB

In packets

Receiving rate
603Pk/s

Received
89525124Pk(s)

Transmitting rate
0Pk/s

Transmitted
3519391Pk(s)

Statistics

Errors while receiving	Errors while transmitting	Packet drops in receiving	Packet drops in transmitting
0 error(s)	0 error(s)	393868Pk(s)	0Pk(s)

lo

Receiving rate
0.00B/s (0Pk/s)

Transmitting rate
0.00B/s (0Pk/s)

126.46MiB received (in **196169Pk(s)**)
0 error(s) faced (**0Pk(s)** dropped)

126.46MiB transmitted (in **196169Pk(s)**)
0 error(s) faced (**0Pk(s)** dropped)

eno1

Receiving rate
206.00B/s (3Pk/s)

Transmitting rate
0.00B/s (0Pk/s)

5.83GiB received (in **5507176Pk(s)**)
0 error(s) faced (**0Pk(s)** dropped)

407.43MiB transmitted (in **2566540Pk(s)**)
0 error(s) faced (**0Pk(s)** dropped)

wlo1

Receiving rate
91.75KiB/s (604Pk/s)

Transmitting rate
0.00B/s (0Pk/s)

15.55GiB received (in **83821779Pk(s)**)

157.93MiB transmitted (in **756682Pk(s)**)

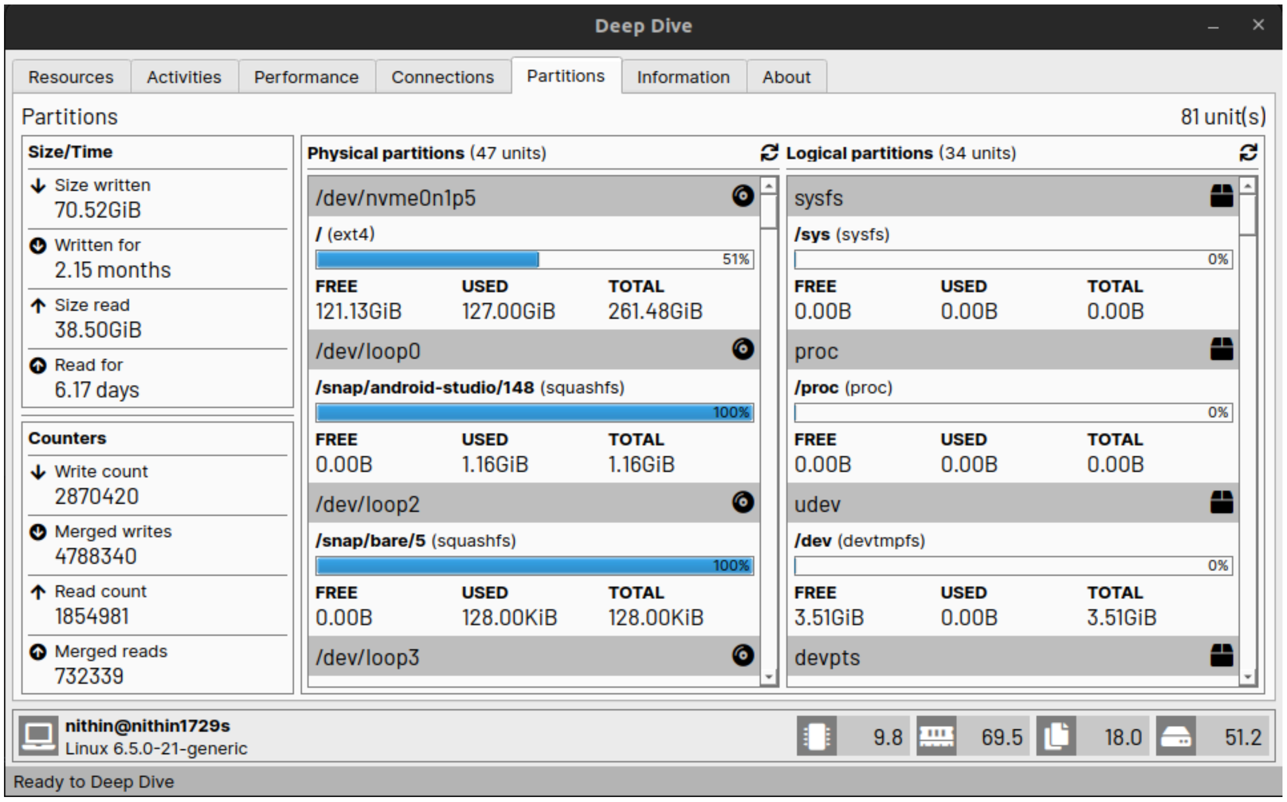
nithin@nithin1729s
Linux 6.5.0-21-generic

9.8 69.5 18.0 51.2

Ready to Deep Dive

Figure 6.4: Connections Tab Screen

Partition Tabscreen : Shows details about the physical and logical partition in the hard disk. It also includes no of read count, write count, merged writes etc



Partitions — 81 unit(s)

Size/Time

Size written
70.52GiB

Written for
2.15 months

Size read
38.50GiB

Read for
6.17 days

Counters

Write count
2870420

Merged writes
4788340

Read count
1854981

Merged reads
732339

Physical partitions (47 units)

/dev/nvme0n1p5
/ (ext4) — 51%

FREE	USED	TOTAL
121.13GiB	127.00GiB	261.48GiB

/dev/loop0
/snap/android-studio/148 (squashfs) — 100%

FREE	USED	TOTAL
0.00B	1.16GiB	1.16GiB

/dev/loop2
/snap/bare/5 (squashfs) — 100%

FREE	USED	TOTAL
0.00B	128.00KiB	128.00KiB

/dev/loop3

Logical partitions (34 units)

sysfs
/sys (sysfs) — 0%

FREE	USED	TOTAL
0.00B	0.00B	0.00B

proc
/proc (proc) — 0%

FREE	USED	TOTAL
0.00B	0.00B	0.00B

udev
/dev (devtmpfs) — 0%

FREE	USED	TOTAL
3.51GiB	0.00B	3.51GiB

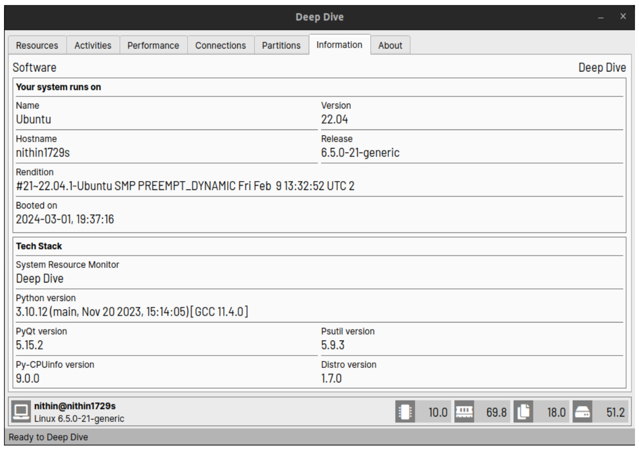
devpts

nithin@nithin1729s
Linux 6.5.0-21-generic
9.8 69.5 18.0 51.2

Ready to Deep Dive

Figure 6.5 : Partitions Tab Screen

Information & About Tabscreen : Gives information regarding the linux distro and the version of dependencies.



Software — Deep Dive

Your system runs on

Name	Version
Ubuntu	22.04
Hostname	**Release**
nithin1729s	6.5.0-21-generic

Rendition
#21~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Fri Feb 9 13:32:52 UTC 2

Booted on
2024-03-01, 19:37:16

Tech Stack

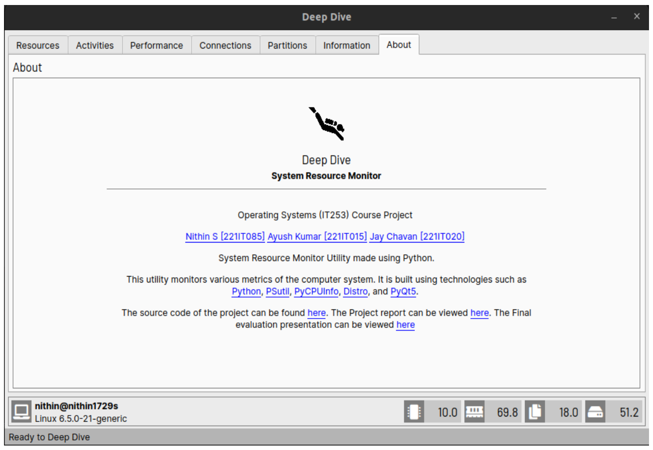
System Resource Monitor
Deep Dive

Python version
3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]

PyQt version	Psutil version
5.15.2	5.9.3
Py-CPUinfo version	**Distro version**
9.0.0	1.7.0

nithin@nithin1729s
Linux 6.5.0-21-generic
10.0 69.8 18.0 51.2

Ready to Deep Dive



About

Deep Dive
System Resource Monitor

Operating Systems (IT253) Course Project

Nithin S [221IT085] Ayush Kumar [221IT015] Jay Chavan [221IT020]

System Resource Monitor Utility made using Python.

This utility monitors various metrics of the computer system. It is built using technologies such as Python, PSutil, PyCPUInfo, Distro, and PyQt5.

The source code of the project can be found here. The Project report can be viewed here. The Final evaluation presentation can be viewed here

nithin@nithin1729s
Linux 6.5.0-21-generic
10.0 69.8 18.0 51.2

Ready to Deep Dive

Figure 6.6 : Information Tab Screen

Figure 6.7: About Tab Screen

FUTURE WORK

Reduce StartUp Time: Come up with a new architecture which reduces the number of function calls
to reduce the start up time of the application. The goal would also be to decrease the overhead of the application.

Enhanced Data Analysis: Introduce advanced statistical analysis and machine learning algorithms to provide deeper insights into system resource usage patterns, enabling predictive analysis for proactive resource management and optimization.

Multi-Platform Support: Extend compatibility to various operating systems beyond the current scope, such as macOS and Linux distributions, to cater to a broader user base and enhance the tool's versatility.

Customization Options: Implement features that allow users to customize the dashboard layout, color schemes, and visualization preferences according to their specific requirements and preferences.

Alerting Mechanisms: Incorporate real-time alerting mechanisms to notify users about critical system resource thresholds being exceeded, enabling timely intervention to prevent performance degradation or system failures.

Historical Data Analysis: Develop functionality to store and analyze historical system resource usage data, facilitating trend analysis, capacity planning, and long-term performance monitoring.

Remote Monitoring and Management: Introduce capabilities for remote monitoring and management, allowing users to monitor system resources across multiple machines from a centralized dashboard and perform management tasks remotely.

Integration with Cloud Platforms: Integrate with popular cloud platforms such as AWS, Azure, and Google Cloud to provide seamless monitoring and management of cloud-based resources, enabling hybrid and multi-cloud environments.

Security and Compliance Features: Enhance the tool's security features to ensure compliance with industry standards and regulations, such as GDPR and HIPAA, by implementing encryption, access controls, and audit trails for sensitive system resource data.

Resource Allocation Optimization: Develop algorithms and tools to dynamically adjust resource allocation based on workload demands and performance metrics, optimizing resource utilization and enhancing overall system efficiency.

REFERENCES

G. Juve et al., "Practical Resource Monitoring for Robust High Throughput Computing," 2015 IEEE International Conference on Cluster Computing, Chicago, IL, USA, 2015, pp. 650-657, doi: 10.1109/CLUSTER.2015.115.

M. Andreolini, M. Colajanni, M. Pietri and S. Tosi, "Real-time adaptive algorithm for resource monitoring," Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013), Zurich, Switzerland, 2013, pp. 67-74, doi: 10.1109/CNSM.2013.6727811