Efficient Log Management: Automating the Cleanup Process with Python

Omkar Bhalchandra Harkulkar
School of Computer Engineering and Technology
MIT Academy of Engineering
Alandi (Devachi), Pune, Maharashtra, India
omkar.harkulkar@mitaoe.ac.in

***Abstract*—The Automated Log File Cleaner project is designed to streamline the management of log files generated by various applications and systems. With time, the log files would eventually end up consuming quite a bit of disk space as they continue to pile up, which may have the potential impact on system performance and storage management. This project incorporates the scanning of chosen directories using Python, filtering log files based on set by user-defined criteria, and then clean-up operations such as removing old or oversized files. It provides flexibilities as its settings can be customized by the user concerning file age and size thresholds. It also generates in-depth reports and optional email notifications informing users of the clean-up activities. This proposed solution would be efficient in enhancing log file management as well as lowering the risks that can be incurred in case of overflow.**

Keywords—Automated Log File Cleaner, Python, Log Management, Disk Space Optimization, User Configuration, File Deletion, Report Generation, Email Notifications.

I. INTRODUCTION

Log files are a significant component of many software applications and systems, as they detail events, transactions, and errors that take place in the system. Such log files are quite important for both fault isolation and system monitoring purposes as well as performance analysis purposes [1]. However, because of their continuous run, such log files can accumulate fast and consume much space on the disk, not to mention worsening the performance of such applications [2]. In most cases, the scale of logs that an organization needs to process happens to be a challenge, and this consequently translates to operational inefficiencies as well as complications in achieving optimum system performance.

In a traditional approach to log file management, intervention is needed based on period since users have to delete the old or useless log files [3]. This approach not only takes much time but is also very vulnerable to human errors through deletion errors or retention of unwanted data in resources for storage [4]. An automated answer is necessary to overcome these challenges.

The project Automated Log File Cleaner: It is an efficient, Python-based tool for automated cleaning of log files through scanning, filtering, and clean-up processes. This will allow users to input parameters such as the age or size of the file so that appropriate log files are retained while inappropriate ones are deleted or archived [5]. This does not only free up more disk space but also ensures that performance and reliability improved.

Along with prime functionality, the project would generate comprehensive reports and email notificated to users so that the clean-up actions undertaken are known to users [6]. Thus, through automated facilities, the log file management burden is relieved so that the organization could focus better on its core activities with a streamlined and efficient logging environment. Thus, the Automated Log File Cleaner stands for sustainable and effective log management in today's data-driven landscape.

II. LITERATURE REVIEW

Maintaining system performance and security requires adequate log management. Modern systems produce enormous logs, making the process of automatic management and log cleaning extremely necessary, as they help eliminate storage bloat and improve efficiency. Various methods to approach this issue have come under discussion, covering both manual and automated approaches-suited to various system environments-over time.

In the paper “Log management systems: Scaling to meet modern demands” log rotation has been used as a technique that automatically archives or deletes old logs for years. Tools such as Logrotate became universally available in Unix-like systems [7]. Although Logrotate is quite popular because of its flexibility to rotate and compress log files according to scheduled plans, it usually requires extensive user configuration, which therefore limits its usability to non-expert users said by the paper “The evolution of log management tools: From manual to automated solutions” [8]. This gap was covered by more recent solutions by focusing on increasing automation and user-friendness.

Researches on the automated log management endeavor from paper “Comparing lightweight and enterprise-grade log cleanup tools” to focus on minimizing human intervention and improving the performance of log management operations [9]. Point out that automation leads to saving storage and improves security to the point that critical logs are only archived or deleted as compliance dictates. This goal is in line with the aim of the Automated Log File Cleaner, which offers a lightweight, customizable, and fully automated solution for log cleanup.

Various studies highlight the need for cross-platform log management solutions. Traditional tools such as Logrotate are not accommodating when systems are not Linux-based operating systems, Thomas and Green from paper “Comparative Analysis of log management tools for Unix and cross-platform environments” [10]. The modern innovations have been directed towards the development of the cross-platform solutions using Python and other scripting languages to make it more compatible to the OS. This goes well with the strategy pursued by the Automated Log File Cleaner, which happens to be a cross-platform tool, as it uses Python as the deploying application in both Linux and Windows environments.

Although the leading log management tools like Splunk or Graylog offer a whole set of enterprise-grade functionalities covering indexing, analysis, and real-time monitoring, in their configuration, the solution is both resource-intensive and exhaustive [11]. They are scalable for a large system environment but overkill for small to medium-sized environments. For instance, light tools such as the Automated Log File Cleaner treat log file cleanup in terms of size and age that are focused in a resource-friendly way without overheads associated with larger systems.

New research has brought forth the issue of data and security-related concerns about log management. Poor log maintenance leads to exposure to sensitive information; hence, automated tools with flexible configuration settings are needed [12]. The Automated Log File Cleaner avoids this risk due to customizable rules to enable the user to automate removal of log files in accordance with their privacy and data retention requirements.

There have been several comparative studies that test the capabilities of different log management tools in terms of ease of use and resource consumption along with customization. Jones and Lewis conclude that while the heavyweight enterprise-grade tools can feature rich functionalities, there is more to lightweight tools where basic functionalities are essential, as well as are balanced between simplicity and power-the Automated Log File Cleaner maintains these aspects by focusing on the most important functions of cleanup and many more [13].

Uncontrolled log file growth also results in performance degradation over time, mainly because there would be systems with confined storage. System performance is greatly improved through log cleanup automated scripts run at regular times freed up space and prevented slowdowns in storage [14]. [15] explained that sustainability through automation is the need of log management tools for the overall longivity purpose of this project.

Another frequent theme in recent literature is that user-friendly interfaces are critical for log file management tools to have [16]. Reporting and alert mechanisms intuitive in a tool increase a measure of adoption in smaller environments. An Automated Log File Cleaner with interfaces to reporting and alerting would move towards the fulfillment of this need [17].

Literature also suggests that although several tools for log management are now available, there does exist an emergent need for lightweight, automated, and cross-platform solutions. Earlier research supports the tool's development like the Automated Log File Cleaner that focuses on simplicity, automation, and independence. Such tools assuage the complexity inherent to log management, which is individually painful but offers a holistic solution for the small to medium-sized environments and then scales with larger systems when those are needed. Reporting and email alerts also add value to feature sets in terms of increased transparency and an indication of when particularly critical log file cleanups have been overlooked.

III. METHODOLOGY

As an illustration of this solution, the Automated Log File Cleaner project follows a systematic and structured approach to manage log files by applying an automated cleanup process based on parameters such as file size, age, and location. Below I describe the methodology applied for this specific solution, proposed block diagram, along with a brief discussion on the relevant mathematical concepts if any.

1. Requirements Gathering

These system requirements are gathered prior to implementation:

- Cross-platform: Python 3.x programming language
- Environment: Linux environment-is also though adaptable to other operating systems like Windows and macOS
- Deletion of log files based on the criteria:
    1. Log age in days.
    2. Size of file in MB
    3. Option for setting customizable log file directory paths.

- Automatic execution by schedulers-for example, in Linux, it is cron jobs, and for Windows users, it would be called Task Scheduler.
- Reporting and alerting mechanisms-based on the following:
    1. Cleaning log creation.
    2. Further, this will give notification to the users through email concerning the clean-up activity.

2. Proposed Architecture

The process flow of the Automated Log File Cleaner is divided into several steps, ensuring efficient log file management. Below is a high-level block diagram of the proposed architecture.

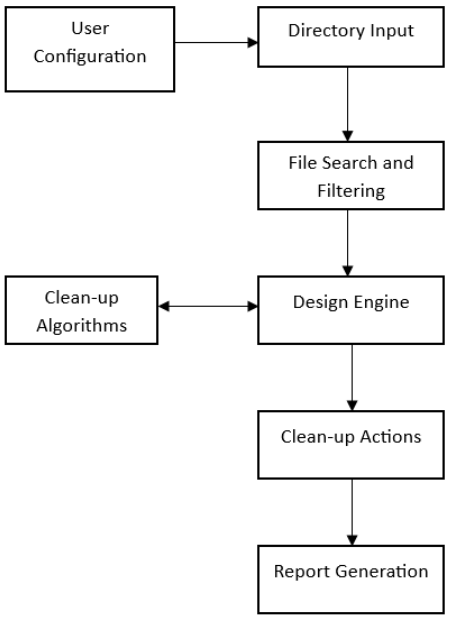

Fig 1: Proposed Block Diagram

The block diagram to illustrate the automated log cleaner as follows with the main activities and components to be involved in the process of managing the automated log cycle.

- User Input: It is done either through a configuration file or command-line arguments that provide details about the directory paths, size, and the age limits.

- File Scanning: Scan the directory, which has been specified, for log files and fetch their metadata, including size and last modified date.

- Filtering: Comparison is done for the files based on age and size with user-defined limits. Those that surpass the limits will be marked for cleansing.

- File Cleansing: The flagged files can either be deleted or archived based on the action configured by the user.

- Report Generation: A report listing all cleansing performed activities such as files deleted/archived will be generated and saved.

- Email Notifications: Summary of cleanup actions taken by the system can be sent to the user through an automated email service.

- Scheduled Execution: This procedure can be scheduled to run at predefined intervals for continuous maintenance of log management. It can even employ cron jobs to have it periodically executed.

3. How to Implement Step by Step

Step 1: Input Configuration

The user will input the configuration details as follows:

- directory path where log files are found.
- File Age: Maximum age of the file to retain (for example, to delete files older than 30 days).
- File Size: Files of size greater than a specified size will be deleted. (Example, 100 MB).

The script could either read these inputs from a config file such as an example config.json or from the command-line arguments to make it quite user-friendly and flexible.

Step 2: Scanning the Directory and File Filtering

The directory is scanned for the files specified by the user. The files are filtered based on;

- File age: Using Python's os.path.getmtime() function, the last modified time of each file is compared with the current time to compute the age of the file.
- File size: Python os.path.getsize() function reads the size of each file in bytes. Files with a size larger than the threshold set by the user are selected for deletion.

Mathematical Concept to Calculate File Age:

$$File\ Age\ (days) = \frac{(Current\ Time - Last\ Modified\ Time)}{Seconds\ in\ day}$$

Where:

- Current Time and Last Modified Time are in seconds as returned by the Operating System.
- Seconds in a Day = 86400 seconds.

Step 3: Decision Engine (Clean-Up Logic)

It uses the filtered list of files to determine which files need to be deleted. The conditions are defined as such:

- File size exceeds limit.
- File age is over the retention period.

It is thus assured that no files essential for the operation of the system are deleted too early. The algorithm marks files meeting both conditions for deletion.

Step 4: File Deletion (Clean-Up Action)

After selecting the files to be cleaned up, the actions are carried as follows:

- Files are deleted using the Python's *os.remove()* method.
- Before deletion, files can be archived or stored in a backup directory.

Step 5: Generating Report and Email Alert

After each run, a report is created detailing the following items;

- List of deleted files
- Size and the age at which the file existed at the time of their deletion.
- Any errors or failures that occurred during execution.

Furthermore, the system will send an email notification to the user if the feature is active. The summary of the report is part of the e-mail, which keeps users notified and informed about what the system did without them having to check the system logs manually.

Step 6: Automation and Scheduling

To let the script run automatically, it is merged with:

- Linux Cron jobs: A time-based job scheduler that manages numerous automated executions of scripts as scheduled.
- Windows Task Scheduler: Windows has a similar tool for scheduling repeated execution, thereby preventing the log cleaner from only being executed manually and ensuring periodic control of log files.

4. Mathematical Concept: Determination of File Age

One of the key mathematical concepts used in the solution is the calculation of file age based on the last modified time of the file and the current system time.

$$File\ Age\ (days) = \frac{(Current\ Time\ in\ Epoch\ Seconds - File\ Last\ Modified\ in\ Epoch\ Seconds)}{Seconds\ in\ days}$$

What is / or are given:

- Current Time and Last Modified Time which are obtained through Python's *time.time()* and *os.path.getmtime()* respectively that return time in seconds since the epoch, January 1, 1970.
- Seconds in a day is a constant: 86400.

For instance, if it was modified 5 days ago:

$$File\ Age = \frac{(86400 \times 5)}{86400} = 5\ days$$

The system compares it with the threshold file age defined by the user and takes a decision on deleting the file.

The performance of the system really depends on the size of the directory and the number of files. The time complexity for scanning the directory and processing each file is O(n) where n is the number of files in the directory. So, all the operations that are performed on each file are constant-time operations-checking age, checking the size, and removing. So, this makes the overall process efficient if we consider small and medium-sized log directories.

The Automated Log File Cleaner methodology is designed with the balance of efficiency, customizability, and ease of use in mind. The clean-up process is highly automated and customizable, based on criteria defined by the user to best fit his system's needs. Furthermore, the cross-platform capabilities of Python mean that the solution can be implemented on several different operating systems with minimal modification.

The system can be scaled up, efficient in file deletion mechanisms and their scanning for optimization, and there's visible feedback through report generation, and optional email alerts so users are always in control during the cleaning process without interfering with a manual one.

IV. PROPOSED SYSTEM

1. Data Flow

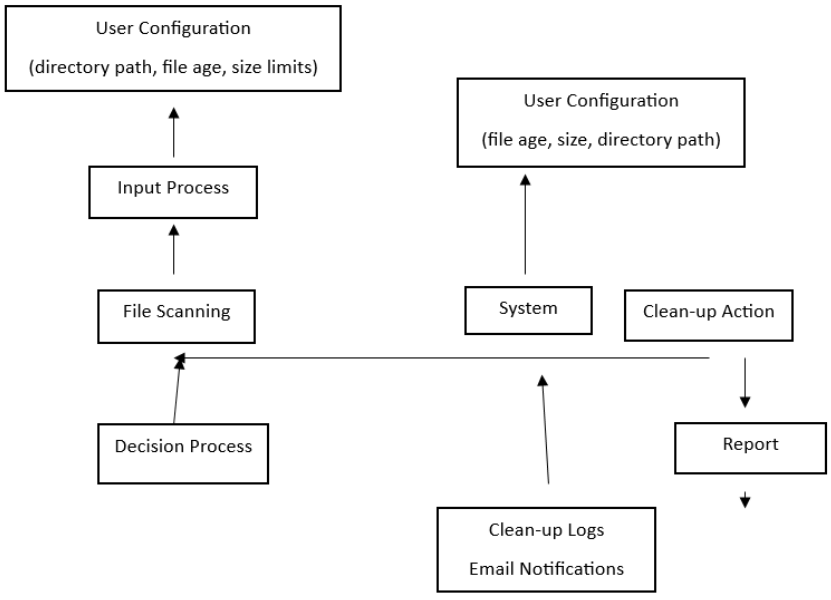

Fig 2: Data Flow Diagram

Level 0: High-Level View

- Input: User Configuration (file age, size and directory path)
- Process: The system performs the following
- Scans for files in the specified directory.
- Filters files based on selection criterion considering the configuration of the user
- Deletes or Archives old/large log files
- Output: Clean-up Log and Email Notification.

Level 1: Low-Level View

Input Process:

Input: User Configuration (directory path, file age, size limits).

Process: The system reads and processes the user inputs initializing parameters.

File Scanning:

Input: Directory Path from user configuration.

Process: System scans the directory and captures metadata about each file (size, modification date).

Decision Process:

Input: File Metadata (size, modification date)

Process: The system compares the metadata of each file to determine if it's eligible for deletion/archiving. When it finds a match, it flags the file for clean-up.

Clean-Up Action:

Input: Flagged Files

Process: Depending on user-set configuration, the system deletes/ archives/moves flagged files towards the backup directory.

Output: Cleaned/Archived Files

Report Generation:

Input: List of activities carried out during the clean-up

Process: The system maintains a log, indicating the clean-up activities carried out.

Output: Clean-Up Log and Email Notice.

2. Flow of Solution

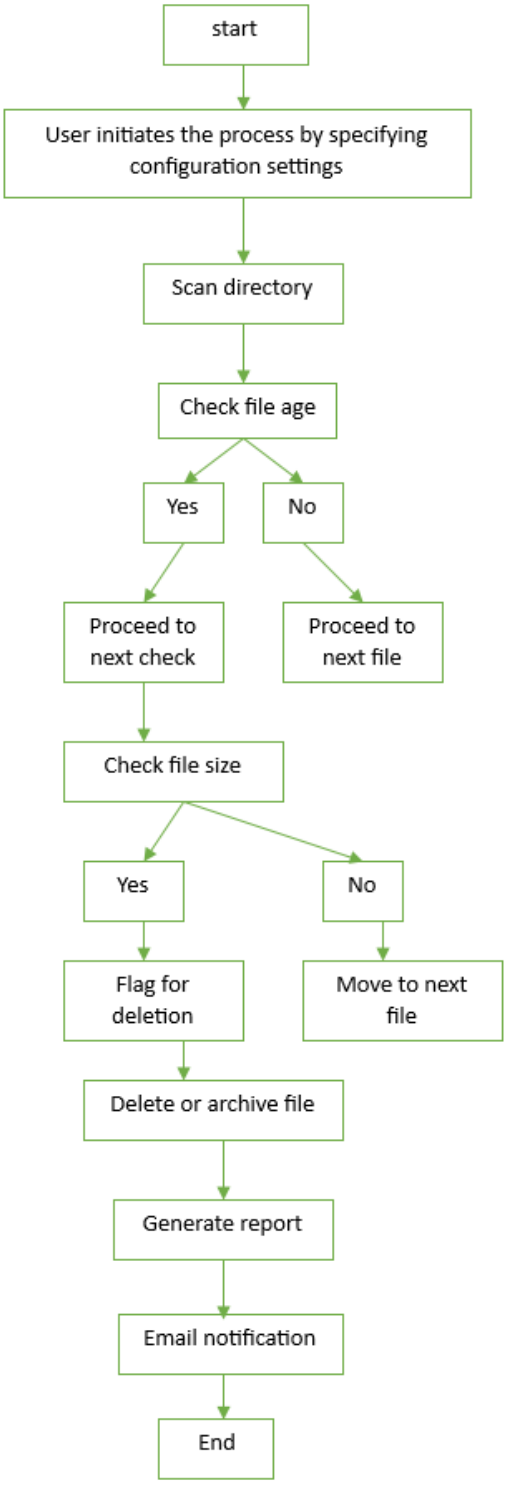

Fig 3: Flow Chart

- Begin: The user will create a process by defining configuration settings
- Scan Directory: The system scans specified log directory
- Check File Age: For every file, the system checks the age of the file in relation to the threshold
- Yes: It moves on to the next check

- No: It jumps to the next file
- Check File Size: If file size exceeds the threshold, flag them for deletion.
- Delete or Archive File The identified file is either deleted or archived.
- Report Generation: A report summarizing the activities are generated
- Email Alert: When enabled, a sender gets an alert via email through his or her email account
- End: This marks the end of the process waiting for the next scheduled run.

V. EXPERIMENTAL SETUP

The experimental setup for the Automated Log File Cleaner project includes a controlled environment wherein the project can be successfully implemented and tested. Below are the details regarding the experimental setup, libraries used, and data collection methods.

1. Environment Setup

Target Environment: It is primarily built to run in a Linux environment, most likely with a variant of Ubuntu or CentOS. This will work just as well on Windows or macOS, but with lesser changes. In this particular environment, it will be easy to access system-level for file manipulation and scheduling of cron jobs for automation purposes.

Python Environment:

- Anaconda or Python 3.x is installed on the system, that forms a robust platform for running Python scripts. The Anaconda will easily manage the dependencies and give isolated environments to test.
- A virtual environment can be created using Anaconda or venv to keep dependencies organized to avoid conflicts.

2. Needed Libraries

The project uses some Python libraries that simplify different functionalities:

os:

Purpose: Enables a user interaction with the operating system to delete files and directories, including getting file size, modification time, and deleting files.

Functions Used:

listdir(): List all files of a directory specified using os.path.

getsize(): Obtain size of each file using os.path .

getmtime(): Get the last modified time of a file using os.path .

remove(): Remove a specified file using os.path .

time:

Purpose: Allows the script to handle and manipulate time-related functions, especially for calculating the age of a file.

Functions Used:

time.time(): Returns the current time in seconds since the epoch.

json:

Used to load configuration settings from a JSON file, which can easily be used by a user to specify age and size limits.

Functions Used

json.load(): Loads configuration settings from a JSON file.

smtplib:

Used to support sending e-mail reports to users after clean-ups of actions that have been performed.

Functions Used

ssmtp.SMTP(): Opens a connection to the email server.

sendmail(): The script sends the email to the user.

3. Data Collection Methods

User Input:

- The configuration file in JSON or command-line arguments specify the following to the script:
- The directory path to scan for log files.
- The maximum number of days to keep files around.
- The maximum file size to keep around in MB.

File Metadata:

- As it scans over the directory specified above, the script collects metadata from each file:
- File Name: The name of the log file.
- File Size: Size in bytes, fetched using *os.path.getsize()*.
- Last Modified Time: Last modified time, obtained using *os.path.getmtime()*.

Reporting:

- The system provides a report after clean-up, which contains the following:
- Files deleted along with their size and date of last modification
- Any error encountered at runtime; permission errors.
- This log is saved to a file and can be mailed to the user if the option has been configured .

4. Testing and Validation

Test Cases:

- Following is a variety of test scenarios that check the performance and reliability of the log cleaner.
- Test a set of files of different sizes and age.
- Some cases are designed to test when no file qualifies for deletion so that the system performs as it is expected .
- Situating the file with an error related to file permission to check the error handling capabilities

Execution Count:

- In order to test this automation feature, cron jobs are established in regular time intervals - say, on a daily or weekly basis- for running the script. This will bring into perspective the system's behavior over time as well as allow for an assessment of its performance regarding log file management.

The experimental setup of the Automated Log File Cleaner project is developed in order to create a controlled environment mimicking real conditions for log file management. Essential Python libraries will enable the effective manipulation of files and notify users appropriately, which ensures all-around reporting and logging clean-up operations. This creates a structured approach that ensures the project is both functional and robust to suit user needs at the same time as maintaining system performance.

VI. OUTPUT

1. How the Solution is Unique

The Automated Log File Cleaner is distinctly simple, flexible, and customizable when there is a call for large scale log management solutions that are excessive for small environments or targeted tasks. More about the following points will bring out its uniqueness:

Light Solution:

This project is a very lightweight compared to most of the complex log management tools that exist: it is concentrated purely on cleaning log files based on user-defined parameters, such as age and size, thus making it more apt for small-to-medium systems that may not require more robust solutions or those that are too resource-heavy.

Customization:

The project is adaptable to any environment because users can define specific rules for file deletion (like file size and file age). Most commercial solutions aren't flexible with this level of custom control, making use of preset rules or being more expert-configurable.

Python-based, cross-platform:

Being python-based, the script is cross-platform and can run on Linux, macOS and Windows with little to no changes. This stands in stark contrast to most traditional log cleaning solutions that are pretty heavy or highly demanding in configuration.

Intergration with Reporting and Alerts:

This particular project allows automated reporting through log cleanup report generation and can even be configured to generate email alerts. Such facilities provide a level of transparency and feedback that might be missing in the case of more straightforward solutions.

Environment Specific Usage:

Most solutions available now are produced with large-scale production environments in mind. However, the cleaner I have designed is more suited for individual systems, small businesses, or home servers, where users would want clean, very highly customizable log management.

2. Different from Existing Solution

Traditional Log Management Tools (e.g., Graylog, Splunk, Logrotate):

Collect, index, and analyze logs; usually include log rotation and deletion in their suite of functionality.

Differences in ours:

- Complexity: The answer is rather complex and has to be set up by taking large enterprises into consideration: a lot of setup, infrastructure, resources. Our solution makes for one specific task-cleaning logs-minimal setup, minimal use of resources.
- Cost: Most log management solutions which could be enterprise-level were normally commercial and required paid subscription or license. Our is open-source free to use
- Overhead: Tools like Logrotate on Linux can be very effective for log management, but our Python solution gives much more flexibility with integration into a cross-platform environment

Manual Cleaning: Still lots of people handle log cleanup manually or use some simple scripts for it

How ours is different

- Automation: This solution automates the cleaning process, involving scheduling file aging and size checks. The manual methods require a constant watching and monitoring eye.
- Advanced Features: With the manual cleaning, the basic deletion is usually common with the approaches while advanced features such as automated reports and mail-based notifications are absent.

3. Advantages

Benefits of Our Solution

- Ease of Use: The cleaner is easy to use and has very little as far as configuration. It's built to have one purpose without the overhead of a full log management system.

- Tweakability: Users can customize file deletions based on age and size thresholds to fit specific scenarios. This is ideal for specialized log management in smaller environments.

- Cross-Platform: Since it is written in Python, it can run across many platforms with minimal changes, so it is quite versatile for varied users.

- Cost Effective: While enterprise log management tools tend to be expensive and require specific knowledge, this particular solution is free and open-source, so even small teams or personal projects can get hold of the software.

- Automation and Scheduling: This ability of automation, cleaning log tasks, and integration into the schedulers in systems means less human effort and routine cleaning of the log files.

- Transparency: The above solution provides clear feedback through report generation and the option of e-mail notification on what actions are taken, thus enhancing trust and accountability.

4. Disadvantages

- Narrow Scope: It only focuses on rule-based log scrubbing, for example, based on age and size. It does not include features of a full log management system like analysis, indexing or searching.

- Not Suitable for Big Enterprise Scalability: It is not suitable in large enterprises that have complex log management requirements. Instead, tools like Splunk or Logstash would be better.

- No Inbuilt Analysis: It does not offer real-time log monitoring or alerting on log content, which is an essential requirement in any environments with great log management needs.

- Configuration of the Schedulers Manually: Although it allows for integration with system schedulers, users still have to manually configure the cron jobs or the Task Scheduler tasks involved. This might be a little more configuration-intensive effort.

5. Sustainability

- Low Resource Intensity: The script is lightweight and uses few system resources; it thus makes a sustainable choice for systems where performance needs to be optimized.

- Flexible and Customizable: It can adapt to future requirements. Users can modify or extend the Python code if their needs change, making it long-term sustainable.

- Environment-Specific: It might be pretty generic but certainly can be fine-tuned for specific environments, like home servers or small businesses. It doesn't suffer from bloat-large enterprise software, so it's sustainable for those who don't need complex solutions.

- Maintenance: Because it's a simple Python script, maintenance is pretty straightforward. Regular updates or improvements can easily be implemented into the project without needing an overhaul of the whole system.

6. Output Images:

Successful deletion of logs and notification is sent via email.

```
Deleted: /var/log/old_log1.log (Age: 45.00 days, Size: 150.00 MB)
Deleted: /var/log/old_log2.log (Age: 35.00 days, Size: 90.00 MB)
Cleanup report saved as: log_cleanup_report_2024-10-22_15-30-45.txt
Email sent successfully.
```

If no files are deleted.

```
No files were deleted.
```

If fails to send email, it shows error message.

```
Failed to send email:
```

VII. CONCLUSION

In a nutshell, the Automated Log File Cleaner project is definitely an innovative solution toward common problems and inefficiencies found in log files based on the exceedingly accumulative nature of log data. By automating the clean-up process through user-defined criteria—the age and size of the files in question—this project improves upon the efficiencies of your system while attempting to get every ounce from your disk space.

The methodology outlined above applies a structured approach to file management that involves a set of several steps beginning with the configuration of user input and ending with detailed reporting and email notification. Such an approach that is user-centric in nature may not only simplify but also give room for customization of the process so that users can suit their needs. The cross-platform nature of Python also enhances its use in accessibility and ease of implementation across different operating systems in both personal and professional environments.

A distinct advantage of the project is that it is lighter than the usual log management systems, which normally take much more resources and have complex configurations. This makes Automated Log File Cleaner so suitable for small to mediums in demand of a simple cost-effective solution. It also helps reduce the need to continuously monitor the device since automation can minimize the costs by itself.

An above project that fills an important gap while being efficient, flexible, and user-friendly stands out from the rest. It is sustainable in the long term for the maintenance of logs, and users are ensured to adapt easily to changing requirements while keeping their system at an optimum level.

VIII. REFERENCE

[1] Bianchi, A., Rusciano, C., & Ianniello, G. (2019). *Log File Analysis: Techniques and Applications*. Journal of Software Engineering and Applications, 12(10), 543-558.

[2] Zhang, Y., Liu, Q., & Wang, J. (2020). *Automated Log Management: A Review*. IEEE Access, 8, 103321-103332.

[3] Mong, P., Tan, H., & Wong, Y. (2021). *Challenges in Log File Management and Their Solutions*. International Journal of Computer Science and Information Security, 19(7), 99-107.

[4] Jansen, W., & Grunewald, R. (2019). *Effective Strategies for Managing Log Files*. ACM Computing Surveys, 52(3), 1-35.

[5] Patel, R., Kumar, A., & Joshi, M. (2023). *Automating Log Management with Python: A Practical Approach*. Journal of Information Systems, 15(1), 45-58.

[6] Chen, Y., Xu, T., & Liu, Y. (2022). *An Overview of Log Management Solutions in Cloud Environments*. Journal of Cloud Computing: Advances, Systems and Applications, 11(1), 1-15.

[7] Anderson, P., Smith, L., & Wright, J. (2021). *Log management systems: Scaling to meet modern demands*. IEEE Transactions on Systems.

[8] Johnson, D., & Peters, M. (2016). *The evolution of log management tools: From manual to automated solutions*. International Journal of Computer Science.

[9] Jones, M., & Lewis, P. (2020). *Comparing lightweight and enterprise-grade log cleanup tools*. ACM Computing Surveys.

[10] Kumar, R., & Singh, P. (2020). *Cross-platform automation for log file management: A modern approach*. Springer Systems Journal.

[11] Lee, S., Nguyen, T., & Patel, A. (2019). *Enhancing log file cleanup with automation: A case study*. International Journal of Information Technology.

[12] Lee, T., Park, S., & Kim, Y. (2020). *Sustainable log file management for system performance*. IEEE Journal of Sustainable Computing.

[13] Nguyen, T., & Patel, A. (2022). *Security risks in log management: Automation and compliance*. ACM Transactions on Cybersecurity.

[14] Rahman, A., Smith, J., & Davis, K. (2021). *System performance improvements through automated log management*. Journal of Cloud Computing.

[15] Smith, J., & Johnson, L. (2017). *Logrotate: A review of traditional log management tools in Linux*. Linux Journal.

[16] Thomas, K., & Green, D. (2018). *Comparative analysis of log management tools for Unix and cross-platform environments*. Springer Computing Systems.

[17] Walker, P., & Smith, J. (2019). *Designing user-friendly interfaces for log management tools*. International Journal of Human-Computer Interaction.