Automated Log File Analysis Website

Bhogalapalli Sandesh

April 2025

Contents

Chapter 1

Introduction

Log files are vital elements in todayś computing environments, supporting performance monitoring, debugging, and security. They provide detailed records of system activities, user behavior, errors, and application usage. Yet, as systems become ever more sophisticated and the amount of data increases, manual log analysis is not just time-consuming but also extremely error-prone and inefficient.
This project aims to construct a strong web-based platform based on Flask for the backend, Bash scripting for log parsing and preprocessing, and Python for dynamic visualization. By parsing, structuring, filtering, and analyzing Apache log files automatically, the platform converts raw, unorganized data into CSV files in structured form and actionable visualizations. Users are able to pull insights quicker, enhancing monitoring, debugging, and decision-making at the operations level.The program then makes it possible for users to filter, sort, and run in-depth analysis on the resultant logs using dynamic tables and visualization.
In addition, automated analysis facilitates users' ability to make data-based decisions through the downloadable visual plots and CSV summaries. The results can also be analyzed further by other external programs such as PowerPoint or Excel, given that offline access to processed datasets is made possible by the platform.

Chapter 2

Requirements

2.1 Software and Modules

- Python 3.13.2
- Bash Shell (Ubuntu)
- Flask (`pip install flask`)
- Matplotlib (`pip install matplotlib`)
- Other modules: `os, subprocess, re`

2.2 Bash Utilities

- sed
- awk
- regex

Chapter 3

Running Instructions

1. Clone or download the project files.
2. Install all Python dependencies.
3. Go to the project directory.
4. Start the Flask server:

   ```
   python app.py
   ```

5. Open a web browser and go to http://127.0.0.1:5000

Chapter 4

Website Layout

4.1 Upload Page

- Upload Apache-style log files (.log).
- Automatic parsing and CSV conversion using Bash scripts.
- Error shown if the file format is incorrect.

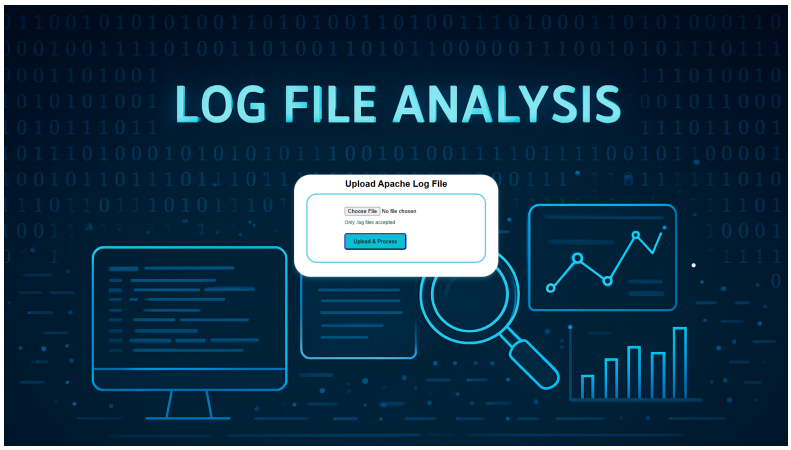

Figure 4.1: Upload Page Interface

4.2 Display Page

- Organized CSV logs in a table.
- Filtering by Event ID and Log Level.
- CSV download feature.
- Filtering within specified time intervals

4.3 Analysis Page

- Line Graph: Events Over Time.
- Pie Chart: Notice vs Error Level Distribution.
- Bar Chart: Event Code Frequency.

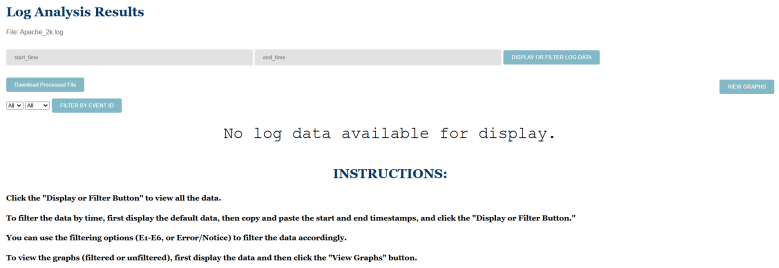

Figure 4.2: Display Page Interface

- Users provide custom Python code.
- Dynamic graph creation.

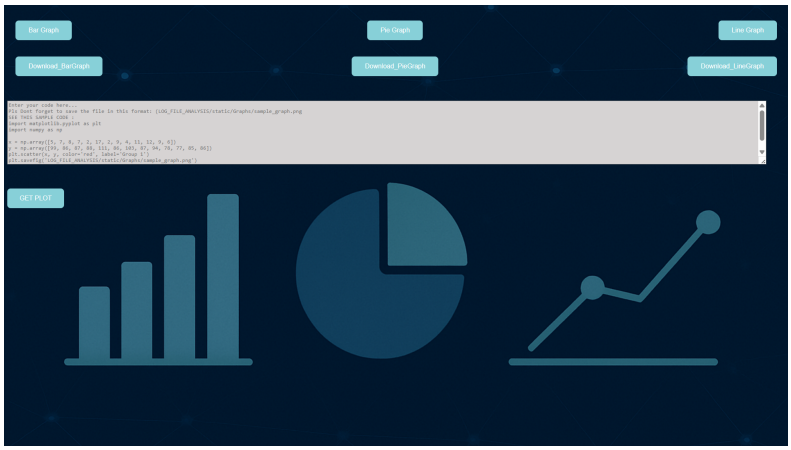

Figure 4.3: Analysis and Graph page Interface

Chapter 5

Modules and Technologies Used

5.1 Flask

Manages routing, user sessions, uploads, and page rendering. Imported Flask, render template, request, session, redirect, url_for, Response, send_from_directory, send_file from flask.

5.2 Matplotlib

Creates pre-defined graphs: line plots, pie charts, and bar graphs. Imported matplotlib.pyplot.

5.3 Subprocess

Runs the Bash scripts from Flask.

5.4 Bash Scripts and Awk

- Parsing of logs.
- Generation of CSV.
- Filtering of events.

Chapter 6

Directory Structure

The project is organized systematically to ensure modularity and clarity. The folder structure is described below:

- `app.py`: Core Flask backend that manages routing, session handling,Bash script execution, and rendering of frontend pages dynamically.
- `static/`: Contains static assets like Bash scripts, stylesheets (CSS), background images, and generated analysis graphs.
  - `bash/`: Collection of Bash scripts responsible for parsing Apache logs, sorting and filtering by Event ID and log level, and time-range-based extraction of entries.
    - `default_display.sh`: Reads the uploaded Apache log file, identifies event patterns, assigns Event IDs, and outputs a full CSV without any filtering.
    - `filtered_display.sh`: Filters the log entries based on a given start and end time range and outputs only the relevant subset of data in structured CSV format.
    - `default_event_sorting.sh`: Sorts all log entries based on Event IDs to easily observe event distribution in the full dataset.
    - `filtered_event_sorting.sh`: Sorts only the filtered logs (within a specified time window) by Event IDs to focus on a narrower subset of events.
    - `default_eventid_filtering.sh`: Extracts entries matching a specific Event ID or log level (notice/error) across the entire log without time filtering.
    - `filtered_eventid_filtering.sh`: Filters log data simultaneously by Event ID, log level, and within a user-specified timestamp range.
  - `css/`: Collection of CSS stylesheets to enhance the visual appeal and usability of different pages.
    - `log_upload_style.css`: Styling for the file upload page, including upload box formatting and buttons.
    - `log_display_style.css`: Styling for the log display table, including table header formatting, alternating row colors, and error message designs.
    - `graphs_style.css`: Manages layout and styling of graph visualization pages.
    - `new_graph_style.css`: Styles the custom graph generation page where users can write and execute Python scripts.
  - `Graphs/`: Folder where dynamically generated graph images are stored after analyzing the logs.
    - `bargraph.png`: Bar graph showing count distribution across Event IDs (E1–E6).
    - `linegraph.png`: Line plot showing the number of events occurring over time.
    - `piegraph.png`: Pie chart showing the ratio of notice and error log levels.
    - `sample_graph.png`: Example plot generated when users run custom Python code.
  - `images/`: Contains background images used in styling various pages.
    - `1.png`: Background image for the upload page.

        * `2.png`: Background image for analysis and graph pages.
    - `js/`: JavaScript files that add interactivity on the frontend (e.g., file upload behavior).
        * `script.js`: JavaScript for managing frontend logic like form submission and basic field validation.

- `templates/`: HTML templates rendered by Flask to dynamically display processed data and graphs.
    - `log_upload.html`: Page where users upload raw log files.
    - `log_display.html`: Displays logs in a structured table with filtering, sorting, and download options.
    - `Graphs.html`: Hub page to navigate to different graphs (bar, line, pie).
    - `bar_graphs.html`: Displays the Event Code Distribution Bar Chart.
    - `line_graphs.html`: Displays Event Count vs Time Line Graph.
    - `pie_graphs.html`: Displays Level State Distribution Pie Chart.
    - `new_graph.html`: Page for writing and running custom graph plotting Python code.
- `uploads/`: Folder where uploaded raw log files are stored temporarily for parsing and analysis. Old files are replaced when new files are uploaded.

Chapter 7

Basic and Advanced Features

7.1 Basic Features

- Upload log files.
- Automatic CSV conversion.
- Download structured CSV.
- Display logs in a table.
- Predefined visualizations (bar, pie, line).
- Time-range filtering.

7.2 Advanced Features

- Dynamically sort and filter table.
- Custom Python code execution embedded.

Chapter 8

Project Journey

8.1 Key Learnings

- Bash scripting integration with Flask.
- Secure handling of user uploads and session management.
- Dealing with too many files and lots of code.

8.2 Challenges

- Gaining expertise in integrating Flask with Bash scripts.
- Parsing anomalous timestamp formats.
- Processing very large log files optimally.
- Debugging and fixing code issues.

8.3 Solutions

- Seeking assistance from tutorials, websites, and friends.
- Extensive format validation using awk.
- Patience and consistent motivation.
- Divide and Conquer method and some ChatGPT.

Bibliography

[1] Flask Web Framework, https://flask.palletsprojects.com/

[2] Loghub Log Datasets, https://github.com/logpai/loghub

[3] Matplotlib Documentation, https://matplotlib.org/stable/

[4] RealPython Flask Tutorial, https://realpython.com/tutorials/flask/

[5] CodePen, https://codepen.io/AmanChourasia/pen/WNpOpXZ

[6] ChatGPT, https://chatgpt.com/,for some queries and background images generation

[7] StackOverflow,
https://stackoverflow.com/questions/55447599/how-to-send-data-in-flask-to-another-page
https://stackoverflow.com/questions/24577349/flask-download-a-file

[8] Google Gemini for comparing Timestamps, https://gemini.google.com/