UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+ ACCREDITED UNIVERSITY

PROJECT REPORT

Course Code- 24CAP-607

LINUX ADMINISTRATION LAB

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

MCA(GENERAL)

Submitted By:

Name: - Satyam Yadav

UID: - 24MCA20393

Branch: - M.C.A(General)

Section: - 24MCA6-B

Submitted To:

Er. Prabhjot Kaur

Assistant Professor

Abstract

This project report presents the development of a system information dashboard for Linux. The dashboard is designed to provide real-time monitoring of system resources such as CPU usage, memory utilization, disk space, and network activity. The aim is to assist system administrators and users in tracking their system's performance and diagnosing potential issues efficiently. This project leverages open-source tools and libraries to collect, store, and visualize system metrics, creating an interactive and user-friendly interface.

Introduction

A system information dashboard is an essential tool for monitoring and managing the performance of a Linux system. As modern systems grow more complex, the need for real-time monitoring solutions becomes critical. Such dashboards help in visualizing the system's health, identifying bottlenecks, and making informed decisions regarding resource allocation and system optimization. This project aims to develop a comprehensive dashboard that consolidates various system metrics into a single, easily accessible interface.

Objectives

1. Develop a web-based dashboard that provides comprehensive monitoring of system resources.
2. Implement real-time data collection and visualization to ensure up-to-date information.
3. Ensure the dashboard is user-friendly and accessible, catering to both novice and advanced users.
4. Provide detailed information on CPU, memory, disk usage, and network activity.
5. Incorporate alerting mechanisms to notify users of critical system events.

Tools and Technologies

1. Python: Utilized as the backend programming language for data collection and processing due to its simplicity and extensive library support.
2. Flask: A lightweight web framework used to build the backend server that handles data retrieval and serves the dashboard.
3. psutil: A Python library used for retrieving information on system utilization (CPU, memory, disks, network, sensors) and system uptime.
4. InfluxDB: A time-series database chosen for storing and querying large amounts of time-stamped data efficiently.
5. Grafana: An open-source platform for monitoring and observability, used to create and display the dashboard with real-time visualizations.

#!/bin/bash # System Information Dashboard # Function to print the dashboard header print_header() { echo "==========" echo " SYSTEM INFORMATION DASHBOARD" echo "==========" } # Function to get CPU usage get_cpu_usage() { echo "CPU Usage:" # Using `top` to get CPU usage. Using `grep` and `awk` to extract CPU data. top -bn1 | grep "Cpu(s)" | sed "s/.*, *\([0-9.]*\)%* id.*/\1/" | awk '{print " CPU Usage: " 100 - $1"%"}' echo } # Function to get Memory usage get_memory_usage() { echo "Memory Usage:" # Using `free -h` to get human-readable memory stats free -h | grep Mem | awk '{print " Used: " $3 " / Total: " $2 " (" $3/$2*100 "%)"}' echo } # Function to get Disk usage get_disk_usage() { echo "Disk Usage:" # Using `df -h` to get disk usage statistics for mounted file systems df -h | grep '^/dev' | awk '{print " " $1 " - " $5 " used, " $3 " free of " $2 " total"}' echo } # Function to get the top 5 CPU-consuming processes get_top_processes() { echo "Top 5 Processes by CPU Usage:" # Using `ps` to get top 5 processes sorted by CPU usage ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%cpu | head -n 6 echo } # Function to get system uptime and load averages get_system_uptime() { echo "System Uptime and Load Averages:" # Using `uptime` to get system uptime and load averages uptime | awk '{print " Uptime: " $3 " " $4 " " $5 "\n Load Averages: " $9 " (1 min), " $10 " (5 min), " $11 " (15 min)"}' echo } # Main function to display the dashboard display_dashboard() { print_header get_cpu_usage get_memory_usage get_disk_usage get_top_processes get_system_uptime } # Execute the dashboard display display_dashboard use this code also

System Information Dashboard in Linux

Project Report

Submitted by RISHAB 24MCA20391

In partial fulfilment for the degree of

MASTER OF COMPUTER APPLICATIONS (MCA)

UNIVERSITY INSTITUTE OF COMPUTING (UIC)

CHANDIGARH UNIVERSITY

Abstract

This project report presents the development of a comprehensive system information dashboard for Linux. The dashboard is designed to provide real-time monitoring of various system resources such as CPU usage, memory utilization, disk space, and network activity. By leveraging open-source tools and libraries, the dashboard aims to assist system administrators and users in tracking their system's performance and diagnosing potential issues efficiently.

Introduction

A system information dashboard is an essential tool for monitoring and managing the performance of a Linux system. As modern systems grow more complex, the need for real-time monitoring solutions becomes critical. Such dashboards help in visualizing the system's health, identifying bottlenecks, and making informed decisions regarding resource allocation and system optimization. This project aims to develop a comprehensive dashboard that consolidates various system metrics into a single, easily accessible interface.

Objectives

1. Develop a web-based dashboard that provides comprehensive monitoring of system resources.
2. Implement real-time data collection and visualization to ensure up-to-date information.
3. Ensure the dashboard is user-friendly and accessible, catering to both novice and advanced users.
4. Provide detailed information on CPU, memory, disk usage, and network activity.
5. Incorporate alerting mechanisms to notify users of critical system events.

Tools and Technologies

1. **Python**: Utilized as the backend programming language for data collection and processing due to its simplicity and extensive library support.
2. **Flask**: A lightweight web framework used to build the backend server that handles data retrieval and serves the dashboard.
3. **psutil**: A Python library used for retrieving information on system utilization (CPU, memory, disks, network, sensors) and system uptime.

4. **InfluxDB**: A time-series database chosen for storing and querying large amounts of time-stamped data efficiently.
5. **Grafana**: An open-source platform for monitoring and observability, used to create and display the dashboard with real-time visualizations.

Implementation

1. Data Collection

The first step in developing the dashboard is setting up a mechanism to collect system metrics. The psutil library is used to gather data on various aspects of system performance, including CPU usage, memory utilization, disk usage, and network activity. The collected data is structured into a time-series format and periodically stored in InfluxDB, ensuring that historical data is preserved for trend analysis.

2. Backend Development

The backend of the dashboard is built using Flask, a micro web framework in Python. Flask handles HTTP requests and serves the frontend application. It also provides APIs for retrieving real-time data from InfluxDB. The backend processes the data collected by psutil and ensures it is available for visualization.

3. Frontend Development

Grafana is used for the frontend development of the dashboard. Grafana connects to InfluxDB and retrieves the stored metrics. It provides a variety of visualization options, such as line graphs, bar charts, and gauges, allowing users to customize their dashboard according to their needs. Grafana's interactive and intuitive interface makes it easy for users to navigate through different metrics and identify performance trends.

4. Integration

Integration of the backend and frontend components is crucial for a seamless user experience. The Flask backend serves the Grafana frontend, making the dashboard accessible via a web browser. This integration ensures that users can interact with real-time data and adjust the dashboard settings as needed. The combined setup allows for an efficient and cohesive monitoring solution.

Features

1. **Real-Time Monitoring**: The dashboard provides up-to-date information on system resources, with data being refreshed at regular intervals.
2. **User-Friendly Interface**: Grafana's intuitive interface ensures that users can easily navigate and understand the displayed metrics.
3. **Customizable Views**: Users can customize the dashboard to display the metrics that are most relevant to their needs, allowing for a personalized monitoring experience.
4. **Alerts and Notifications**: Configurable alerts notify users of critical system events, such as high CPU usage or low disk space, enabling proactive management.
5. **Historical Data Analysis**: The time-series data stored in InfluxDB allows users to analyze historical trends and make informed decisions about system performance and capacity planning.

Challenges

1. **Data Collection**: Ensuring accurate and timely data collection from various system components is a significant challenge. The psutil library must be configured correctly to provide reliable metrics without introducing additional overhead.
2. **Performance Optimization**: Both the backend and frontend components must be optimized to handle large amounts of data efficiently. This involves tuning database queries, minimizing latency, and ensuring that the dashboard remains responsive under heavy load.
3. **Security**: Protecting sensitive system information from unauthorized access is crucial. The dashboard must implement robust authentication and authorization mechanisms to ensure that only authorized users can access the data.
4. **Scalability**: As the number of monitored systems increases, the dashboard must scale to handle the additional load. This requires efficient resource management and potentially distributing the workload across multiple servers.

Script Implementation

```
# System Information Dashboard

# Function to print the dashboard header

print_header() {

   echo
"================================================"

   echo "       SYSTEM INFORMATION DASHBOARD"

   echo
"================================================"

}

# Function to get CPU usage

get_cpu_usage() {

   echo "CPU Usage:"

   # Using `top` to get CPU usage. Using `grep` and `awk` to extract
CPU data.

   top -bn1 | grep "Cpu(s)" | sed "s/.*, *\([0-9.]*\)%* id.*/\1/" | awk
'{print "  CPU Usage: " 100 - $1"%"}'

   echo }

# Function to get Memory usage

get_memory_usage() {

   echo "Memory Usage:"

   # Using `free -h` to get human-readable memory stats

   free -h | grep Mem | awk '{print "  Used: " $3 " / Total: " $2 " ("
$3/$2*100 "%)"}

  echo

}

# Function to get Disk usage

get_disk_usage() {

   echo "Disk Usage:"

   # Using `df -h` to get disk usage statistics for mounted file systems
```

```
    df -h | grep '^/dev' | awk '{print "  " $1 " - " $5 " used, " $3 " free of
" $2 " total"}'
    echo
}
# Function to get the top 5 CPU-consuming processes
get_top_processes() {
    echo "Top 5 Processes by CPU Usage:"
    # Using `ps` to get top 5 processes sorted by CPU usage
    ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%cpu | head -n 6
    echo
}
# Function to get system uptime and load averages
get_system_uptime() {
    echo "System Uptime and Load Averages:"
    # Using `uptime` to get system uptime and load averages
    uptime | awk '{print "  Uptime: " $3 " " $4 " " $5 "\n  Load
Averages: " $9 " (1 min), " $10 " (5 min), " $11 " (15 min)"}'
    echo
}
# Main function to display the dashboard
display_dashboard() {
    print_header
    get_cpu_usage
    get_memory_usage
    get_disk_usage
    get_top_processes
    get_system_uptime
}
# Execute the dashboard display
display_dashboard
```

Implementation

1. Data Collection

The first step in developing the dashboard is setting up a mechanism to collect system metrics. The psutil library is used to gather data on various aspects of system performance, including CPU usage, memory utilization, disk usage, and network activity. The collected data is structured into time-series format and periodically stored in InfluxDB, ensuring that historical data is preserved for trend analysis.

2. Backend Development

The backend of the dashboard is built using Flask, a micro web framework in Python. Flask handles HTTP requests and serves the frontend application. It also provides APIs for retrieving real-time data from InfluxDB. The backend processes the data collected by psutil and ensures it is available for visualization.

3. Frontend Development

Grafana is used for the frontend development of the dashboard. Grafana connects to InfluxDB and retrieves the stored metrics. It provides a variety of visualization options, such as line graphs, bar charts, and gauges, allowing users to customize their dashboard according to their needs. Grafana's interactive and intuitive interface makes it easy for users to navigate through different metrics and identify performance trends.

4. Integration

Integration of the backend and frontend components is crucial for a seamless user experience. The Flask backend serves the Grafana frontend, making the dashboard accessible via a web browser. This integration ensures that users can interact with real-time data and adjust the dashboard settings as needed. The combined setup allows for an efficient and cohesive monitoring solution.

Features

1. Real-Time Monitoring: The dashboard provides up-to-date information on system resources, with data being refreshed at regular intervals.
2. User-Friendly Interface: Grafana's intuitive interface ensures that users can easily navigate and understand the displayed metrics.
3. Customizable Views: Users can customize the dashboard to display the metrics that are most relevant to their needs, allowing for a personalized monitoring experience.
4. Alerts and Notifications: Configurable alerts notify users of critical system events, such as high CPU usage or low disk space, enabling proactive management.
5. Historical Data Analysis: The time-series data stored in InfluxDB allows users to analyze historical trends and make informed decisions about system performance and capacity planning.

Challenges

1. Data Collection: Ensuring accurate and timely data collection from various system components is a significant challenge. The psutil library must be configured correctly to provide reliable metrics without introducing additional overhead.
2. Performance Optimization: Both the backend and frontend components must be optimized to handle large amounts of data efficiently. This involves tuning database queries, minimizing latency, and ensuring that the dashboard remains responsive under heavy load.
3. Security: Protecting sensitive system information from unauthorized access is crucial. The dashboard must implement robust authentication and authorization mechanisms to ensure that only authorized users can access the data.
4. Scalability: As the number of monitored systems increases, the dashboard must scale to handle the additional load. This requires efficient resource management and potentially distributing the workload across multiple servers.

Conclusion

The system information dashboard developed in this project provides a comprehensive tool for monitoring and managing Linux systems. By leveraging open-source tools like Python, Flask, psutil, InfluxDB, and Grafana, the dashboard offers real-time insights into system performance. It helps users identify and address performance issues, optimize resource utilization, and ensure the overall health of their systems. As technology evolves, future enhancements such as AI-driven analytics, advanced alerting mechanisms, and integration with other monitoring tools will further enhance the dashboard's capabilities.