Automated Package Update and Health Check

Software Engineering Assignment

Submitted by

Manthan Sharma – 23BCS11533

Maheshwar – 23BCS11580

Harsh Jha – 23BCS12103

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

Chandigarh University

January - May 2025

Automated Package Update and Health Check

1. Objective:

The goal of this assignment is to create an effective automation system by combining the concepts of software engineering design with real-world Linux usage. The system will run health checks and automate package updates in a Linux-based environment. Students will use shell scripting and Linux system utilities to develop a solid solution by putting concepts like modularity, cohesion, coupling, abstraction, encapsulation, and design modelling into practice.

2. Introduction:

To guarantee security, stability, and peak performance, Linux systems need to be maintained on a regular basis. Two essential components of system upkeep are:

- avoiding vulnerabilities by keeping software packages updated.
- keeping an eye on system health to spot possible problems before they get out of hand.

Managing these duties by hand can be laborious and prone to mistakes. In accordance with software engineering principles, this project suggests an automated method that uses shell scripts to carry out package upgrades and health checks.

The system is going to:

- Installing and recognizing package updates should be automated.
- Keep an eye on important system parameters including CPU and memory consumption, storage space availability, and service status.
- Create thorough reports that summarize the outcomes of the health check and update actions.
- Inform administrators of important problems or successful activities by email or other means.

3. Problem Statement:

3.1 Manual Maintenance Difficulties

Large-scale system maintenance is frequently difficult for Linux administrators:

- Security patches need direct involvement, but regular package updates are essential.
- Monitoring system health entails routinely examining a number of parameters, including CPU, memory, and disk utilization.
- Manual procedures are prone to inefficiency and human mistake.

3.2 Automation Is Necessary

These issues can be resolved by automation by:

- lowering the amount of human labour required for routine chores like package upgrades.
- ensuring that system problems are promptly identified through health checks.
- delivering trustworthy notifications and reports so that decisions are well-informed.

The goal of this project is to create an automated system that meets these criteria by being scalable, modular, and reusable.

4. Advanced Features

Software package updates and system health monitoring are the two primary functions of the Automated Package Update and Health Check system. These tasks are broken down into straightforward, intelligible functionalities:

4.1 Management of Package Updates

The Linux system's installed software packages are all kept up to date thanks to this feature.

1. Look for any updates:

- The package manager (apt, yum, or dnf, for example) is used by the system to automatically check for available updates.
- This is accomplished by a scheduled task (cron job) at regular intervals, like every day.

2. Set up updates:

- The system installs updates automatically if there are any available.
- The user does not need to manually enter any information; the operation is silent.

3. Record Update Activities:

- All update operations, whether successful or unsuccessful, are documented in a log file.
- For auditing purposes, the log contains information such as package names, versions, and timestamps.

4.2 Monitoring of System Health

This feature keeps an eye on the system's operation and spots possible problems.

1. Keep an eye on important metrics:

The system verifies crucial metrics like:

- CPU Usage: Makes sure there isn't too much strain on the processor.
- Use of Memory: Prevents memory fatigue.
- Disk Space: Guarantees there is enough storage.
- Service Status: Verifies that vital services, such web servers, are operating correctly.

2. Alerts and Thresholds:

- A configuration file has predefined boundaries for every measure.
- The system indicates a problem whenever a metric surpasses its upper limit, such as when CPU use over 80%.

3. Record Health Information:

- For reference, the outcomes of health checks are kept in a log file.
- These logs can be used by administrators to monitor patterns and spot reoccurring problems.

4.3 Notification and Reporting

This feature notifies administrators of significant occurrences and provides a summary of the outcomes of package updates and health checks.

1. Create Reports:

   Following upgrades and health checks, the system generates a report. Included in the report are:

- a list of the most recent packages.
- An overview of health indicators.
- Any problems found or suggestions for additional measures.

2. Send Alerts:

   Administrators receive notifications from the system based on the report. Among the notifications are:

- Verification that the modifications were successful.
- notifications of serious health problems (such inadequate disk space).
- recommendations for manual assistance if required.

3. Methods of Notification:

- Depending on the configuration settings, notifications can be sent by SMS, email, or messaging apps like Slack.

This condensed version keeps all the important details while concentrating on the system's functionality and avoiding technical jargon.

5. Software Design Principles

Several software engineering concepts are used in the Automated Package Update and Health Check system's design to guarantee that the final product is effective, scalable, and maintainable.

5.1 Abstraction

Abstraction simplifies complex systems by hiding unnecessary details from users and exposing only essential functionalities:

- Each component of the system is designed as a separate module that performs a specific function:

  - package_updater.sh: Manages all aspects of package updating.
  - health_checker.sh: Handles monitoring of system health metrics.
  - report_generator.sh: Compiles results into a human-readable format.
  - notifier.sh: Sends alerts based on report content.

By abstracting these functionalities, administrators can interact with a straightforward interface without needing to understand the underlying complexities.

5.2 Encapsulation

Encapsulation is the process of combining related information and features into discrete units, or modules, while limiting access to certain parts:

Every script summarizes its capabilities:
- Package_updater.sh, for instance, has routines for installing and determining whether updates are available, but it keeps its internal operations hidden from other scripts.

Each script can separately maintain its state thanks to this encapsulation, which also offers unambiguous interfaces for interacting with other modules.

5.3 Modularity

A design idea known as modularity stresses the division of a system into more controllable, smaller components.

Each of the software's separate modules, or scripts, is in charge of carrying out a certain task:
- Package Updater: In charge of installing and verifying package updates.
- Health Checker: Tracks metrics related to system performance.
- Report Generator: Creates reports by compiling output from various scripts. Notifier: Notifies users based on the results of reports.

Reusability is improved by this modular design, which allows individual scripts to be utilized in many projects or situations without requiring changes.

5.4 Cohesion & Coupling

Whereas coupling describes how dependent modules are on one another, cohesion describes how closely tied and focused a module's responsibilities are:

High Cohesion

Every module serves a clear purpose:

- For instance, health_checker.sh does not manage packages; instead, it concentrates solely on tracking health indicators.

Maintainability is enhanced by high module cohesiveness since modifications to one module are less likely to impact other modules.

Low Cohesion

Instead of using direct dependence, Low Coupling Modules communicate using defined input/output formats (such as logs or JSON files):

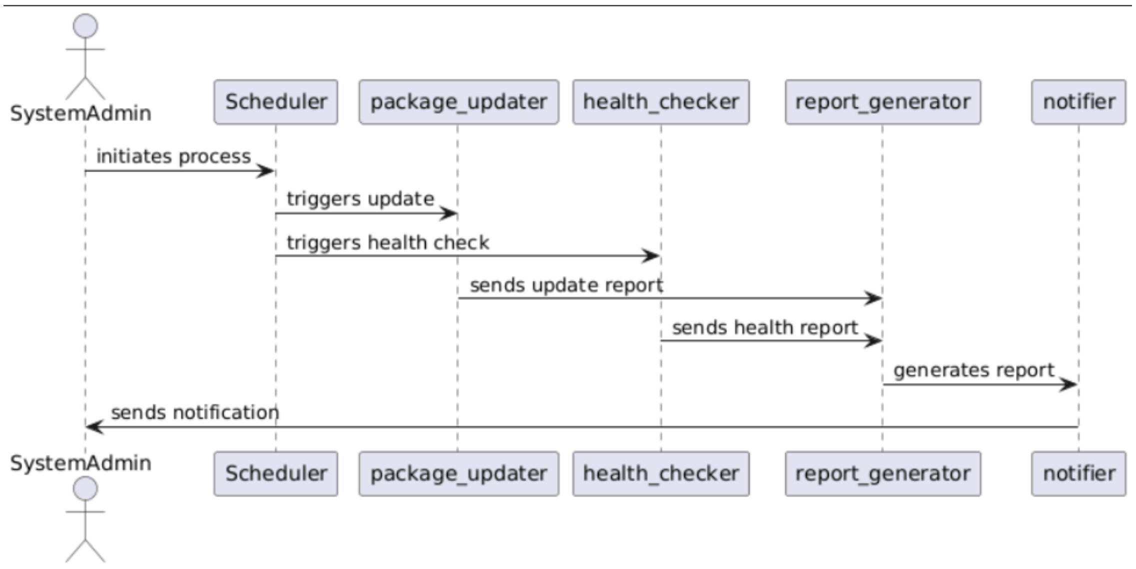
- For instance, report_generator.sh reads output files generated by both package_updater.sh and health_checker.sh.

Because of reduced coupling, developers can change or swap out specific modules without affecting the functionality of the entire system.

6. <u>System Architecture</u>

6.1 Data Flow Diagram (DFD)

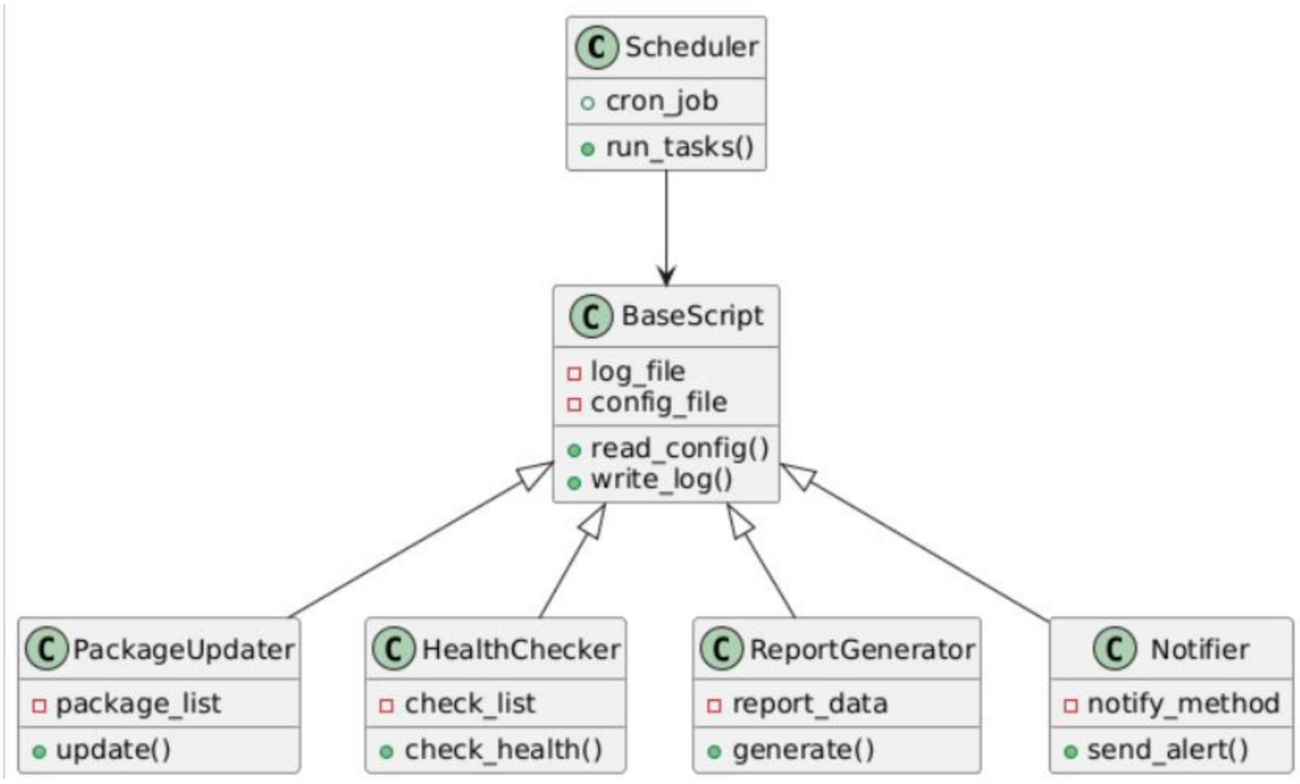
The following diagram illustrates the flow of data within the system:



Justification:

1. The scheduler uses cron jobs, for example, to initiate the package updater and health checker scripts on a regular basis.
2. The report generator receives the outputs from these scripts.
3. The notifier script receives a summary report from the report generator.
4. Depending on the information in the report, the notifier notifies or alerts the system administrator.

6.2 Class Diagram



7. Deployment Design

7.1 Installation Instructions

1. **Clone the Repository:**

- Begin by cloning the project repository from a version control system (e.g., GitHub). This will download all necessary files to your local machine.

```
git clone https://github.com/username/auto-update-health-check.git
cd auto-update-health-check
```

2. **Set Up Configuration:**

- Copy the example configuration file to create a new configuration file that will be customized for your environment.

```
cp config.example.yml config.yml
```

- Open the config.yml file in a text editor (e.g., nano, vim, or gedit) and modify the settings according to your system requirements.

3. **Make Scripts Executable:**

- Ensure that all shell scripts in the project directory are executable. This is necessary for the scripts to run properly.

```
chmod +x *.sh
```

4. **Set Up Cron Job:**

- To automate the execution of the scripts, set up a cron job. This will allow the system to run the main script at specified intervals (e.g., daily).

```
crontab -e
```

  - Add a line to schedule the job. For example, to run the script daily at 2 AM:

```
0 2 * * * /path/to/auto-update-health-check/main.sh
```

5. **Install Required Dependencies:**

- Ensure that any necessary dependencies are installed on your system. This may include utilities required for sending notifications or performing health checks.

```
 sudo apt-get update
 sudo apt-get install -y curl jq mailutils  # Adjust based on your
notification method.
```

6. **Verify Permissions and Access:**

- Ensure that the user account running the cron job has sufficient permissions to execute scripts and access necessary files and directories.

7. **Test the System:**

- After deployment, manually run each script to ensure they function as expected.

```
./package_updater.sh
./health_checker.sh
./report_generator.sh
./notifier.sh
```

8. **Monitor Logs:**

- Review log files generated by the scripts to confirm that updates and health checks are being performed correctly.

9. **Adjust Configuration as Needed:**

- Based on initial tests and monitoring, adjust configuration settings in config.yml as necessary.

10. **Documentation and Training:**

- Document installation and operational procedures for future reference.

8. Implementation Details

Overview of Scripts:

- **package_updater.sh:** Uses the configured package management (apt, yum, etc.) to find available updates and installs them covertly.

- **health_checker.sh:** CPU, memory, disk space, and service status are all tracked by health_checker.sh, which compares metrics to thresholds set in config.yml.

- **report_generator.sh:** Aggregates results from the updater and checker scripts into a summary report (e.g., JSON or text format).

- **notifier.sh:** Sends notifications depending on report contents via email or other ways (e.g., Slack webhook).

9. Conclusion

An important development in Linux environment management is the Automated Package Update and Health Check system. This technology improves operating efficiency and adds to the system's overall security and dependability by automating routine maintenance operations.

Key Contributions

1. **Streamlined Maintenance:** System administrators save time and effort by not having to manually intervene when package updates are automated. Instead of getting weighed down by routine upgrades, this enables IT staff to concentrate on more strategic duties.

2. **Enhanced Security:** Updating software packages on a regular basis is essential for safeguarding computers against attacks and vulnerabilities. The method greatly lowers the danger of security breaches that can result from out-of-date software by making sure that all installed packages are current.

3. **Active Health Monitoring:** Critical system indicators are continuously monitored by the health monitoring feature. The system may proactively detect possible problems before they become serious ones by monitoring CPU consumption, memory utilization, storage space, and service status. This proactive strategy aids in preserving uptime and peak performance.

4. **Effective Reporting:** Administrators can gain important insights from the creation of comprehensive reports that summarize update actions and health metrics. By showing patterns, spotting reoccurring problems, and offering suggestions for additional steps, these reports help people make well-informed decisions.

5. **Timely Notifications:** Administrators are guaranteed to be swiftly notified of significant occurrences, such as successful updates or urgent health alarms, thanks to the notification mechanism. This instant feedback loop increases system reliability by enabling administrators to intervene promptly when needed.

Future Enhancements

Even while the existing setup offers a strong basis for automated maintenance, there is room for improvement in the future:

1. **Integration with Configuration Management technologies:** Even more advanced management capabilities may be possible by integrating technologies such as Ansible or Puppet.

2. **Advanced Analytics:** By using past data trends, machine learning algorithms may be able to forecast probable errors.

3. **User Interface Development:** Developing a web-based dashboard may offer a more intuitive way to manage setups and keep an eye on system status.

Final Thoughts

To sum up, the Automated Package Update and Health Check system is a crucial component of contemporary Linux setups. This system increases productivity, security, and dependability by using automation to manage repetitive operations. Using such automated solutions will be essential to sustaining strong IT infrastructures as businesses depend more and more on technology to run their operations.

This project addresses issues that system administrators experience in the real world while providing a hands-on application of software engineering fundamentals. All things considered, this project demonstrates the value of automation in IT administration and lays the groundwork for upcoming advancements in system maintenance procedures.