Software Architecture Document (SAD)

1. Problem Statement

System administrators often struggle with tracking critical errors buried within massive system logs. Manually scanning log files is inefficient and prone to human error, especially during peak server load times. This system automates the detection of serious log entries and instantly notifies the administrator.

2. Objective

The goal is to design and implement a shell-script-based automation system that monitors system logs in real-time or at regular intervals, filters for critical events using pattern matching, and sends alerts to a designated email address. This enhances reliability, reaction speed, and uptime for system-critical services.

3. Software Design Principles Applied

- Abstraction: High-level functionality centers on monitoring logs and notifying users.
- Encapsulation: Alerting and logging are encapsulated in a utility script.
- Modularity: Code is split across logically independent scripts (`log_monitor.sh`, `utils.sh`).
- Cohesion: Each function focuses on a single task, enhancing clarity and testability.
- Coupling: Low coupling through simple function interfaces reduces dependencies.

4. Data Flow Diagram (DFD)

Below is a conceptual Data Flow Diagram:

[Syslog] [Monitor Script] [Filter Critical Logs] [Log to File] + [Send Email Alert]

5. Module Design

The system is divided into modules:

- `log_monitor.sh`: Reads syslog, loops through it line-by-line, applies filters.
- `utils.sh`: Contains `log_message` and `send_alert` functions.
- `test/`: Includes automated test cases using `bats`.
- `setup.sh`: Bootstraps the environment by setting execution permissions.

This structure ensures easy maintenance and testing.

6. Deployment Design

To deploy the system:

1. Clone the repository from GitHub.

2. Give execute permissions: `chmod +x *.sh`

3. Optionally add the script to `crontab` for periodic checks:

```
   */5 * * * * /path/to/log_monitor.sh
```

Dependencies:

- Linux-based system

- Bash

- `mailutils` installed

7. Risk Identification and Mitigation

Risks:

- Log file permission denied Mitigation: Use sudo and set proper file permissions.

- Email configuration not working Mitigation: Pre-check mailutils installation and test email send.

- Script fails on unknown log format Mitigation: Use regex to generalize matching.

- Log flooding Mitigation: Implement rate limiting or alert thresholds.

8. Testing Strategy

Tools used:

- `shellcheck`: For static analysis of Bash scripts.

- `bats`: Bash Automated Testing System for unit testing.

- `time`, `top`, `htop`: To monitor runtime performance.

Tests:

- Verify if log entries with keywords are correctly identified.

- Ensure email alerts are only triggered for relevant lines.

- Simulate log entries and test real-time responses.