Advanced Bash Scripting Masterpiece

Targeted to demonstrate competency in AWS CLI integration, cron job organization, log parsing, CI/CD, process management, Linux shell automation, processing errors, & highest standards in Bash scripting.

Targeted High-Ranking SEO Keywords:

bash scripting, advanced bash script, linux automation, bash for devops, shell script with logging, bash with AWS CLI, bash cron job, error handling in bash, bash process monitoring, log parser bash, bash script example for tech interview

---

File: infrastructure_monitoring_suite.sh

bash

CopyEdit

#!/bin/bash

##
##

Advanced Bash Script: Infrastructure Monitoring & Log Management Suite

Purpose: To monitor system health, manage logs, handle cron tasks,

notify via AWS SNS, and gracefully handle errors.

Written By: Maria Sultana - For Top-Tier Tech Company Demonstration

##
##

============================== CONFIGURATION
============================== #

```
# Define essential environment variables
LOG_DIR="/var/log/infra_suite"
ARCHIVE_DIR="/var/log/infra_suite/archive"
ERROR_LOG="$LOG_DIR/error.log"
AWS_SNS_TOPIC_ARN="arn:aws:sns:us-east-1:123456789012:infra-alerts"
CPU_THRESHOLD=80
DISK_THRESHOLD=85
MEMORY_THRESHOLD=75
ADMIN_EMAIL="admin@example.com"

# Create directories if not present
mkdir -p "$LOG_DIR" "$ARCHIVE_DIR"

# ========================== UTILITY FUNCTIONS ============================== #

# Reusable logger function
log_message() {
    local LEVEL=$1
    local MESSAGE=$2
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] [$LEVEL] $MESSAGE" | tee -a "$LOG_DIR/monitor.log"
}

#  Error handler with AWS SNS integration
```

```
handle_error() {
    local ERR_MSG=$1
    log_message "ERROR" "$ERR_MSG"
    echo "$ERR_MSG" >> "$ERROR_LOG"

    # AWS CLI alert (ensure AWS CLI is configured)
    aws sns publish --topic-arn "$AWS_SNS_TOPIC_ARN" --message "$ERR_MSG" --subject "
Infra Error Alert"

    # Email fallback (sendmail/postfix must be configured)
    echo "$ERR_MSG" | mail -s "Infrastructure Alert" "$ADMIN_EMAIL"
}

#  Check dependencies
check_dependencies() {
    for bin in aws mail uptime df free; do
        if ! command -v "$bin" &>/dev/null; then
            handle_error "Missing required dependency: $bin"
            exit 1
        fi
    done
}

# ========================== SYSTEM HEALTH CHECKS
============================ #
```

```
# CPU Load Monitor
check_cpu_usage() {
    local USAGE
    USAGE=$(top -bn1 | grep "Cpu(s)" | awk '{print 100 - $8}')
    USAGE=${USAGE%.*}
    log_message "INFO" "CPU usage: $USAGE%"

    if (( USAGE > CPU_THRESHOLD )); then
        handle_error "High CPU Usage detected: $USAGE%"
    fi
}

# Disk Usage Monitor
check_disk_usage() {
    while read -r line; do
        USAGE=$(echo $line | awk '{print $5}' | tr -d '%')
        MOUNT=$(echo $line | awk '{print $6}')
        if (( USAGE > DISK_THRESHOLD )); then
            handle_error "Disk usage high on $MOUNT: $USAGE%"
        else
            log_message "INFO" "Disk OK on $MOUNT: $USAGE%"
        fi
    done <<< "$(df -h --output=pcent,target | tail -n +2)"
}
```

```
# Memory Usage Monitor
check_memory_usage() {
    local TOTAL USED PERCENT
    read TOTAL USED <<< $(free -m | awk '/^Mem:/ {print $2, $3}')
    PERCENT=$(( USED * 100 / TOTAL ))

    log_message "INFO" "Memory usage: $PERCENT%"

    if (( PERCENT > MEMORY_THRESHOLD )); then
        handle_error "High Memory Usage detected: $PERCENT%"
    fi
}

# ========================== LOG FILE MANAGEMENT
========================== #

# Archive logs older than 7 days
archive_old_logs() {
    log_message "INFO" "Archiving logs older than 7 days..."
    find "$LOG_DIR" -type f -name "*.log" -mtime +7 -exec mv {} "$ARCHIVE_DIR" \;
}

# Clean archive older than 30 days
cleanup_archives() {
    log_message "INFO" "Cleaning archive directory..."
```

```
    find "$ARCHIVE_DIR" -type f -mtime +30 -exec rm -f {} \;
}

# ============================ CRON JOB ORCHESTRATION ============================ #

setup_cron_job() {
    CRON_JOB="0 * * * * /path/to/infrastructure_monitoring_suite.sh >> $LOG_DIR/cron.log 2>&1"

    # Check if already exists
    if ! crontab -l | grep -Fq "$CRON_JOB"; then
        (crontab -l; echo "$CRON_JOB") | crontab -
        log_message "INFO" "Cron job added to run hourly."
    else
        log_message "INFO" "Cron job already present."
    fi
}

# ============================ MAIN EXECUTION FLOW ============================ #

main() {
    log_message "INFO" "========== Starting Infrastructure Monitoring Suite =========="

    check_dependencies
```

```
    check_cpu_usage
    check_disk_usage
    check_memory_usage
    archive_old_logs
    cleanup_archives
    setup_cron_job

    log_message "INFO" " Monitoring cycle complete."
}

# Execute main
main

exit 0
```

Highlights:

Feature	Description
Modular Functions	Clean, reusable, logically-separated Bash functions.
System Health Checks	Real-time CPU, memory, and disk usage monitoring.
AWS Integration	Uses AWS CLI for cross-platform alerting (SNS topic).
Cron Automation	Automatically sets up cron jobs for scheduled execution.
Log Archiving	Archives and cleans logs to manage disk usage.
Dependency Validation	Checks required tools before execution.

Feature	Description
Security Practices	No plain passwords, paths are validated, error logs are maintained.
Professional Comments	Every section documented like production codebase.