Universal Health Monitor - Configuration File Based

Download Script universal-health-monitor.sh

? One-Line Download & Execute

```
clear && curl -fsSL https://docs.greenhome.stream/attachments/26 -o universal-health-monitor.sh && chmod +x universal-health-monitor.sh && clear && curl -fsSL https://docs.greenhome.stream/attachments/27 -o monitor-domains.txt && chmod +x /root/universal-health-monitor.sh && (crontab -l 2>/dev/null | grep -v "/root/universal-health-monitor.sh"; echo "*/5 * * * *    /root/universal-health-monitor.sh") | crontab - && ./universal-health-monitor.sh
```

Script - Universal Health Monitor - Configuration File Based

A completely universal solution that reads from a configuration file, so users never need to touch the code.

1. Main Script: universal-health-monitor.sh

Create the script file `universal-health-monitor.sh`

```
nano /root/universal-health-monitor.sh
```

Add the below code by simply copy and paste it inside the file `universal-health-monitor.sh`

```
#!/bin/bash

# Universal Multi-Domain Multi-Machine Health Monitor Script
# This script reads domain configurations from monitor-domains.txt and monitors accordingly

# Configuration
CONFIG_FILE="monitor-domains.txt"
SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
LOG_FILE="$SCRIPT_DIR/vm-health-monitor.log"
TIMEOUT=10
MAX_RETRIES=3
```

```
STOP_START_DELAY=5

# Function to log messages with timestamp
log_message() {
    echo "$(date '+%Y-%m-%d %H:%M:%S') - $1" | tee -a "$LOG_FILE"
}

# Function to check if URL is responding
check_url() {
    local url="$1"
    local timeout="$2"

    # Add http:// if not present
    if [[ ! "$url" =~ ^https?:// ]]; then
        url="http://$url"
    fi

    # Use curl with options for both HTTP and HTTPS
    if curl -s -k -L --connect-timeout "$timeout" --max-time "$timeout" "$url" > /dev/null
2>&1; then
        return 0  # Success
    else
        return 1  # Failure
    fi
}

# Function to check if VM/LXC is running
check_machine_status() {
    local machine_id="$1"
    local machine_type="$2"

    case "$machine_type" in
        "vm")
            if qm status "$machine_id" | grep -q "status: running"; then
                return 0  # Running
            else
                return 1  # Not running
            fi
            ;;
        "lxc")
```

```
            if pct status "$machine_id" | grep -q "status: running"; then
                return 0  # Running
            else
                return 1  # Not running
            fi
            ;;
        *)
            log_message "ERROR: Unknown machine type '$machine_type'"
            return 1
            ;;
    esac
}

# Function to stop machine
stop_machine() {
    local machine_id="$1"
    local machine_type="$2"
    local domain="$3"

    log_message "Attempting to stop $machine_type $machine_id for domain $domain"

    case "$machine_type" in
        "vm")
            if qm stop "$machine_id"; then
                log_message "✓ VM $machine_id stop command executed successfully"

                # Wait and verify it stopped
                local wait_count=0
                while [ $wait_count -lt 30 ]; do  # Wait up to 30 seconds
                    if ! check_machine_status "$machine_id" "$machine_type"; then
                        log_message "✓ VM $machine_id confirmed stopped"
                        return 0
                    fi
                    sleep 1
                    ((wait_count++))
                done

                log_message "⚠ VM $machine_id stop command executed but status verification
timed out"
                return 0  # Command succeeded even if verification timed out
```

```
            else
                log_message "✗ Failed to execute stop command for VM $machine_id"
                return 1
            fi
            ;;
        "lxc")
            if pct stop "$machine_id"; then
                log_message "✓ LXC $machine_id stop command executed successfully"

                # Wait and verify it stopped
                local wait_count=0
                while [ $wait_count -lt 30 ]; do  # Wait up to 30 seconds
                    if ! check_machine_status "$machine_id" "$machine_type"; then
                        log_message "✓ LXC $machine_id confirmed stopped"
                        return 0
                    fi
                    sleep 1
                    ((wait_count++))
                done

                log_message "⚠ LXC $machine_id stop command executed but status verification
timed out"
                return 0  # Command succeeded even if verification timed out
            else
                log_message "✗ Failed to execute stop command for LXC $machine_id"
                return 1
            fi
            ;;
    esac
}

# Function to start machine
start_machine() {
    local machine_id="$1"
    local machine_type="$2"
    local domain="$3"

    log_message "Attempting to start $machine_type $machine_id for domain $domain"

    case "$machine_type" in
```

```
        "vm")
            if qm start "$machine_id"; then
                log_message "✓ VM $machine_id start command executed successfully"

                # Wait and verify it started
                local wait_count=0
                while [ $wait_count -lt 60 ]; do  # Wait up to 60 seconds for start
                    if check_machine_status "$machine_id" "$machine_type"; then
                        log_message "✓ VM $machine_id confirmed running"
                        return 0
                    fi
                    sleep 1
                    ((wait_count++))
                done

                log_message "⚠ VM $machine_id start command executed but status verification
timed out"
                return 0  # Command succeeded even if verification timed out
            else
                log_message "✗ Failed to execute start command for VM $machine_id"
                return 1
            fi
            ;;
        "lxc")
            if pct start "$machine_id"; then
                log_message "✓ LXC $machine_id start command executed successfully"

                # Wait and verify it started
                local wait_count=0
                while [ $wait_count -lt 60 ]; do  # Wait up to 60 seconds for start
                    if check_machine_status "$machine_id" "$machine_type"; then
                        log_message "✓ LXC $machine_id confirmed running"
                        return 0
                    fi
                    sleep 1
                    ((wait_count++))
                done

                log_message "⚠ LXC $machine_id start command executed but status verification
timed out"
```

```
                return 0  # Command succeeded even if verification timed out
            else
                log_message "✗ Failed to execute start command for LXC $machine_id"
                return 1
            fi
            ;;
    esac
}

# Function to restart machine
restart_machine() {
    local machine_id="$1"
    local machine_type="$2"
    local domain="$3"

    log_message "=== INITIATING RESTART SEQUENCE FOR $domain ($machine_type $machine_id) ==="

    # Stop the machine
    if stop_machine "$machine_id" "$machine_type" "$domain"; then
        log_message "Waiting $STOP_START_DELAY seconds before starting..."
        sleep "$STOP_START_DELAY"

        # Start the machine
        if start_machine "$machine_id" "$machine_type" "$domain"; then
            log_message "=== RESTART SEQUENCE COMPLETED SUCCESSFULLY FOR $domain ==="
            return 0
        else
            log_message "=== RESTART SEQUENCE FAILED AT START PHASE FOR $domain ==="
            return 1
        fi
    else
        log_message "=== RESTART SEQUENCE FAILED AT STOP PHASE FOR $domain ==="
        return 1
    fi
}

# Function to parse config line
parse_config_line() {
    local line="$1"
```

```
    # Remove leading/trailing whitespace
    line=$(echo "$line" | sed 's/^[[:space:]]*//;s/[[:space:]]*$//')

    # Skip empty lines and comments
    if [[ -z "$line" ]] || [[ "$line" =~ ^# ]]; then
        return 1
    fi

    # Parse format: 'domain','type','id'
    if [[ "$line" =~ ^\'([^\']+)\',\'([^\']+)\',\'([^\']+)\'$ ]]; then
        PARSED_DOMAIN="${BASH_REMATCH[1]}"
        PARSED_TYPE="${BASH_REMATCH[2]}"
        PARSED_ID="${BASH_REMATCH[3]}"
        return 0
    else
        log_message "ERROR: Invalid line format: $line"
        return 1
    fi
}

# Function to validate config file
validate_config_file() {
    local config_file="$1"
    local errors=0
    local line_number=0

    if [[ ! -f "$config_file" ]]; then
        log_message "ERROR: Configuration file '$config_file' not found"
        return 1
    fi

    log_message "Validating configuration file: $config_file"

    while IFS= read -r line; do
        ((line_number++))

        # Skip empty lines and comments
        if [[ -z "$(echo "$line" | sed 's/^[[:space:]]*//;s/[[:space:]]*$//')" ]] || [[
"$line" =~ ^[[:space:]]*# ]]; then
            continue
```

```
        fi

        if parse_config_line "$line"; then
            # Validate domain
            if [[ -z "$PARSED_DOMAIN" ]]; then
                log_message "ERROR: Empty domain at line $line_number"
                ((errors++))
            fi

            # Validate type
            if [[ "$PARSED_TYPE" != "vm" ]] && [[ "$PARSED_TYPE" != "lxc" ]]; then
                log_message "ERROR: Invalid machine type '$PARSED_TYPE' at line $line_number
(must be 'vm' or 'lxc')"
                ((errors++))
            fi

            # Validate ID
            if [[ ! "$PARSED_ID" =~ ^[0-9]+$ ]]; then
                log_message "ERROR: Invalid machine ID '$PARSED_ID' at line $line_number (must
be numeric)"
                ((errors++))
            fi
        else
            if [[ ! "$line" =~ ^[[:space:]]*# ]] && [[ -n "$(echo "$line" | sed
's/^[[:space:]]*//;s/[[:space:]]*$//')" ]]; then
                log_message "ERROR: Invalid format at line $line_number: $line"
                ((errors++))
            fi
        fi
    done < "$config_file"

    if [ $errors -gt 0 ]; then
        log_message "Configuration validation failed with $errors errors"
        return 1
    else
        log_message "Configuration validation passed"
        return 0
    fi
}
```

```
# Function to monitor single domain
monitor_domain() {
    local domain="$1"
    local machine_type="$2"
    local machine_id="$3"

    log_message "Monitoring $domain -> $machine_type $machine_id"

    local retry_count=0
    local domain_failed=true

    # Retry loop for this domain
    while [ $retry_count -lt $MAX_RETRIES ]; do
        if check_url "$domain" "$TIMEOUT"; then
            log_message "✓ $domain is responding (attempt $((retry_count + 1))/$MAX_RETRIES)"
            domain_failed=false
            break
        else
            retry_count=$((retry_count + 1))
            if [ $retry_count -lt $MAX_RETRIES ]; then
                log_message "⚠ $domain not responding (attempt $retry_count/$MAX_RETRIES) -
retrying in 5 seconds..."
                sleep 5
            fi
        fi
    done

    if [ "$domain_failed" = true ]; then
        log_message "✗ $domain failed after $MAX_RETRIES attempts"
        log_message "Initiating restart for $machine_type $machine_id due to $domain failure"

        if restart_machine "$machine_id" "$machine_type" "$domain"; then
            log_message "Successfully restarted $machine_type $machine_id for $domain"
            return 0
        else
            log_message "Failed to restart $machine_type $machine_id for $domain - manual
intervention required"
            return 1
        fi
    else
```

```
        log_message "✓ $domain is healthy - no action needed for $machine_type $machine_id"
        return 0
    fi
}

# Main function
main() {
    log_message "========================================"
    log_message "Starting Universal Multi-Domain Health Monitor"
    log_message "Script directory: $SCRIPT_DIR"
    log_message "Configuration file: $CONFIG_FILE"
    log_message "Log file: $LOG_FILE"
    log_message "========================================"

    # Check if config file exists in script directory
    local config_path="$SCRIPT_DIR/$CONFIG_FILE"

    if [[ ! -f "$config_path" ]]; then
        log_message "ERROR: Configuration file not found at $config_path"
        log_message "Please create $CONFIG_FILE in the same directory as this script"
        exit 1
    fi

    # Validate configuration
    if ! validate_config_file "$config_path"; then
        log_message "Exiting due to configuration errors"
        exit 1
    fi

    local total_domains=0
    local successful_operations=0
    local failed_operations=0

    # Process each line in config file
    while IFS= read -r line; do
        # Skip empty lines and comments
        if [[ -z "$(echo "$line" | sed 's/^[[:space:]]*//;s/[[:space:]]*$//')" ]] || [[
"$line" =~ ^[[:space:]]*# ]]; then
            continue
        fi
```

```
        if parse_config_line "$line"; then
            ((total_domains++))
            log_message "----------------------------------------"

            if monitor_domain "$PARSED_DOMAIN" "$PARSED_TYPE" "$PARSED_ID"; then
                ((successful_operations++))
            else
                ((failed_operations++))
            fi
        fi
    done < "$config_path"

    log_message "========================================"
    log_message "Health check completed"
    log_message "Total domains monitored: $total_domains"
    log_message "Successful operations: $successful_operations"
    log_message "Failed operations: $failed_operations"
    log_message "========================================"

    if [ $failed_operations -gt 0 ]; then
        log_message "WARNING: $failed_operations operation(s) failed - manual intervention may
be required"
        exit 1
    else
        log_message "All monitored services are healthy or successfully restarted"
        exit 0
    fi
}

# Check if running as root
if [ "$EUID" -ne 0 ]; then
    echo "This script must be run as root (required for Proxmox qm/pct commands)"
    exit 1
fi

# Check dependencies
command -v curl >/dev/null 2>&1 || {
    echo "Error: curl is not installed. Please install curl first."
    exit 1
```

```
}

command -v qm >/dev/null 2>&1 || {
    echo "Error: qm command not found. This script must run on a Proxmox host."
    exit 1
}

command -v pct >/dev/null 2>&1 || {
    echo "Error: pct command not found. This script must run on a Proxmox host."
    exit 1
}

# Run main function
main

exit 0
```

2. Configuration File: monitor-domains.txt

The user only needs to edit this file - no code touching required!

```
# Monitor Domains Configuration File
# Format: 'domain','type','id'
#
# domain: The domain or IP to monitor (with or without http://)
# type: Either 'vm' or 'lxc'
# id: The numeric ID of the VM or LXC container
#
# Examples:
'www.domain-1.com','vm','101'
'www.domain-2.com','vm','102'
'www.domain-3.com','lxc','103'
'www.domain-4.com','lxc','104'
'192.168.1.100:8080','vm','105'
'api.example.com','lxc','106'
'https://secure.domain.com','vm','107'

# You can add comments like this
# 'disabled.domain.com','vm','108'

# Multiple entries are supported
```

```
'service1.local','vm','201'
'service2.local:9000','lxc','202'
```

Key Features

 User-Friendly Configuration

- Zero code modification needed
- Simple text file editing
- Clear format with examples and comments
- Automatic validation with helpful error messages

 Flexible Domain Support

- Supports domains with or without protocol (domain.com or https://domain.com)
- IP addresses with ports (192.168.1.100:8080)
- Automatic http:// addition if missing

 Complete Status Verification

- Executes qm stop XXX / pct stop XXX
- Verifies the machine actually stopped
- Waits 5 seconds (configurable delay)
- Executes qm start XXX / pct start XXX
- Verifies the machine actually started

 Comprehensive Logging

- Log file created in same directory as script: vm-health-monitor.log
- Timestamped entries
- Status indicators (✓, ✗, ⚠)
- Detailed restart sequences
- Success/failure tracking

Installation & Usage

1. Setup Files

Place both files in the same directory (e.g., /root/)

```
/root/universal-health-monitor.sh
/root/monitor-domains.txt
```

2. Configure Your Domains

Edit monitor-domains.txt:

```
nano monitor-domains.txt
```

Add your domains using the exact format:

```
'your-domain.com','vm','101'
'192.168.1.50:8080','lxc','103'
'https://api.mysite.com','vm','105'
```

3. Make Executable & Test

```
chmod +x universal-health-monitor.sh
sudo ./universal-health-monitor.sh
```

4. Automate with Cron

```
crontab -e
```

Add the below line at the end of the file:

```
*/5 * * * * /root/universal-health-monitor.sh
```

Configuration Format Rules

? Valid Formats:

```
'domain.com','vm','101'
'192.168.1.100:8080','lxc','102'
'https://secure.com','vm','103'
'www.example.com','lxc','104'
```

? Invalid Formats:

```
domain.com,vm,101              # Missing quotes
'domain.com','VM','101'        # Wrong case (must be lowercase)
'domain.com','vm',101          # ID not quoted
'','vm','101'                  # Empty domain
'domain.com','server','101'    # Invalid type (must be 'vm' or 'lxc')
```

Sample Log Output

```
2025-08-03 18:45:01 - Starting Universal Multi-Domain Health Monitor
2025-08-03 18:45:01 - Configuration validation passed
```

```
2025-08-03 18:45:01 - ----------------------------------------
2025-08-03 18:45:01 - Monitoring www.domain-1.com -> vm 101
2025-08-03 18:45:02 - ✓ www.domain-1.com is responding (attempt 1/3)
2025-08-03 18:45:02 - ✓ www.domain-1.com is healthy - no action needed for vm 101
2025-08-03 18:45:02 - ----------------------------------------
2025-08-03 18:45:02 - Monitoring www.domain-2.com -> vm 102
2025-08-03 18:45:05 - ⚠ www.domain-2.com not responding (attempt 1/3) - retrying in 5
seconds...
2025-08-03 18:45:15 - ✗ www.domain-2.com failed after 3 attempts
2025-08-03 18:45:15 - === INITIATING RESTART SEQUENCE FOR www.domain-2.com (vm 102) ===
2025-08-03 18:45:15 - Attempting to stop vm 102 for domain www.domain-2.com
2025-08-03 18:45:18 - ✓ VM 102 stop command executed successfully
2025-08-03 18:45:19 - ✓ VM 102 confirmed stopped
2025-08-03 18:45:24 - Attempting to start vm 102 for domain www.domain-2.com
2025-08-03 18:45:26 - ✓ VM 102 start command executed successfully
2025-08-03 18:45:35 - ✓ VM 102 confirmed running
2025-08-03 18:45:35 - === RESTART SEQUENCE COMPLETED SUCCESSFULLY FOR www.domain-2.com ===
```

This solution is completely universal - users only need to edit the simple text configuration file, never touching any code!

Download the files

Script universal-health-monitor.sh

Attachment link : universal-health-monitor.sh

Text file monitor-domains.txt

Attachment link : monitor-domains.txt

Ins0mniA

Revision #18
Created 2025-08-03 18:47:30 EEST by Green
Updated 2025-09-04 02:07:01 EEST by Green