Process Monitor Service - Ghid pentru Dezvoltatori

Structura Codului

Componente Principale

Process Monitor Service constă din următoarele fișiere:

1. **monitor_service.sh**: Script-ul principal al serviciului
2. **config.ini**: Fișier de configurare
3. **monitor_service.service**: Definiția serviciului Systemd
4. **setup.sql**: Script de configurare a bazei de date în mediul de testare
5. **test_alarm.sh**: Script interactiv de testare

Organizarea Script-ului

Script-ul `monitor_service.sh` este organizat în secțiuni funcționale:

1. **Gestionarea Configurației**: Funcții pentru citirea și validarea configurației
2. **Logging**: Funcții pentru jurnalizare și rotația jurnalelor
3. **Interacțiunea cu Baza de Date**: Funcții pentru interogarea și actualizarea bazei de date
4. **Gestionarea Proceselor**: Funcții pentru repornirea proceselor folosind diferite strategii
5. **Implementarea Circuit Breaker**: Funcții pentru gestionarea modelului circuit breaker
6. **Bucla Principală**: Bucla principală de monitorizare care leagă totul împreună

Funcții Cheie

Gestionarea Configurației

`read_config()`

Citește și validează configurația din fișierul config.ini.

```
# Read configuration from config.ini
read_config() {
    # Validate config file exists
    # Parse database section
    # Parse monitor section
    # Parse logging section
    # Set defaults for missing values
    # Validate required values
    # Verify numeric values
}
```

`get_process_config()`

Recuperează configurația specifică procesului cu revenire la valorile implicite.

```
# Get process-specific configuration
get_process_config() {
    local process_name="$1"
    local param="$2"
    local default_value="$3"

    # Try process-specific setting
    # Fall back to default process settings
    # Fall back to provided default
}
```

Jurnalizare (Logging)

`log()`

Scrie un mesaj de log cu timestamp, rotește logul dacă este necesar. Mesajele de nivel **ERROR/CRITICAL** sunt trimise și către syslog.

```
log() {
    local level="$1"
    local message="$2"
    local timestamp=$(date '+%Y-%m-%d %H:%M:%S')

    # Check if log rotation is needed
    if check_log_rotation; then
        rotate_logs
    fi

    # Format: YYYY-MM-DD HH:MM:SS [LEVEL] - Message
    echo "$timestamp  [$level] - $message" | tee -a "$LOG_FILE"

    # If level is ERROR or higher, also log to syslog
    if [[ "$level" == "ERROR" || "$level" == "CRITICAL" ]]; then
        logger -p daemon.err "$timestamp [$level] $message"
    fi
}
```

`check_log_rotation()`

Verifică dacă este necesară rotația jurnalelor pe baza dimensiunii fișierului.

```
check_log_rotation() {
    # Check if log file exists
    # Get file size
    # Compare with maximum size
}
```

`rotate_logs()`

Rotește fișierele jurnal când se atinge dimensiunea maximă.

```
rotate_logs() {
    # Remove oldest log file if it exists
    # Rotate existing log files
    # Rotate current log file
    # Create new empty log file
    # Log rotation event
}
```

Interacțiunea cu Baza de Date

`create_mysql_config()`

Creează un fișier de configurare MySQL temporar pentru acces securizat la baza de date.

```
create_mysql_config() {
    # Create empty file
    # Set secure permissions
    # Write MySQL configuration
}
```

`get_alarm_processes()`

Interoghează baza de date pentru procesele în stare de alarmă.

```
get_alarm_processes() {
    # Query database for processes with alarma=1 and sound=0
    # Return process_id, process_name, alarma, sound, notes
}
```

`update_alarm_status()`

Actualizează baza de date pentru a șterge starea de alarmă după repornirea cu succes.

```
update_alarm_status() {
    local process_id="$1"
    # Update database to set alarma=0 and add restart note
}
```

Gestionarea Proceselor

`get_restart_strategy()`

Determină strategia de repornire pentru un proces.

```
get_restart_strategy() {
    local process_name="$1"
    # Get restart strategy from configuration
}
```

`restart_process()`

Încearcă să repornească un proces folosind strategia configurată.

```
restart_process() {
    local process_name="$1"
    # Get restart strategy, max attempts, and restart delay
    # Execute pre-restart command if configured
    # Attempt restart using the appropriate strategy
    # Perform health check after restart
}
```

`execute_pre_restart()`

Execută o comandă pre-repornire dacă este configurată.

```
execute_pre_restart() {
    local process_name="$1"
    # Get pre-restart command from configuration
    # Execute command if configured
}
```

`perform_health_check()`

Verifică repornirea cu succes folosind comanda de health check configurată.

```
perform_health_check() {
    local process_name="$1"
    # Get health check command and timeout
    # Replace %s with process name if present
    # Try health check with timeout and retries
}
```

Implementarea Circuit Breaker

`check_circuit_breaker()`

Verifică dacă circuit breaker-ul este deschis pentru un proces.

```
check_circuit_breaker() {
    local process_name="$1"
    # Initialize if not exists
    # Check if circuit breaker is open
    # Reset if enough time has passed
}
```

`update_circuit_breaker()`

Actualizează starea circuit breaker-ului pe baza succesului/eșecului repornirii.

```
update_circuit_breaker() {
    local process_name="$1"
    local success="$2"
    # Increment failure count on failure
    # Open circuit breaker if failure threshold reached
    # Reset failure count on success
}
```

Modele de Design

Modelul Circuit Breaker

Serviciul implementează modelul circuit breaker pentru a preveni încercările excesive de repornire pentru procesele care eșuează:

1. **Stare Închisă**: Operare normală, încercările de repornire sunt permise
2. **Stare Deschisă**: După mai multe eșecuri, încercările de repornire sunt blocate
3. **Mecanism de Resetare**: Circuit breaker-ul se resetează automat după o perioadă de timp configurată

Detalii de implementare:

```
# Circuit breaker implementation
declare -A circuit_breaker
declare -A failure_counts
declare -A last_failure_times

check_circuit_breaker() {
    # Implementation details
}

update_circuit_breaker() {
```

```
    # Implementation details
}
```

Modelul Strategy

Serviciul utilizează un model strategy pentru repornirea proceselor, suportând mai multe strategii de repornire:

1. **service**: Utilizează systemd pentru a reporni procesul ca serviciu
2. **process**: Oprește și repornește direct procesul
3. **auto**: Încearcă mai întâi repornirea ca serviciu, apoi revine la repornirea ca proces

Detalii de implementare:

```
restart_process() {
    local process_name="$1"
    local strategy=$(get_restart_strategy "$process_name")

    case "$strategy" in
        "service")
            # Service restart strategy
            ;;
        "process")
            # Process restart strategy
            ;;
        "auto"|*)
            # Auto restart strategy
            ;;
    esac
}
```

Considerații de Securitate

Credențiale Bază de Date

Serviciul gestionează credențialele bazei de date în mod securizat:

1. Credențialele sunt citite din fișierul config.ini
2. Un fișier temporar cu permisiuni securizate este creat pentru autentificarea MySQL
3. Fișierul temporar este curățat când scriptul se încheie

```
# Temporary MySQL config file
MYSQL_TEMP_CONFIG=$(mktemp)

# Function to create MySQL config file
create_mysql_config() {
    # Implementation details
}
```

```
# Function to clean up temporary MySQL config file
cleanup_mysql_config() {
    # Implementation details
}

# Set up trap to clean up on exit
trap cleanup_mysql_config EXIT
```

Permisiuni Fișiere

Serviciul necesită permisiuni specifice pentru fișiere:

1. Scriptul în sine ar trebui să fie executabil doar de către root: `chmod 700`
2. Fișierul config.ini ar trebui să fie citibil doar de către root: `chmod 600`
3. Fișierul de jurnal ar trebui să fie inscriptibil de către serviciu, dar citibil de către alții: `chmod 644`

Gestionarea Erorilor

Serviciul implementează o gestionare robustă a erorilor:

1. **Validarea Configurației**: Validează toți parametrii de configurare
2. **Erori de Conexiune la Baza de Date**: Înregistrează erorile și continuă operațiunea
3. **Eșecuri de Repornire**: Implementează circuit breaker pentru a preveni încercările excesive de repornire
4. **Eșecuri de Health Check**: Înregistrează eșecurile și consideră repornirea nereușită

Extinderea Serviciului

Adăugarea de Noi Strategii de Repornire

Pentru a adăuga o nouă strategie de repornire:

1. Adăugați un nou caz în funcția `restart_process()`
2. Implementați logica de repornire pentru noua strategie
3. Actualizați documentația pentru a descrie noua strategie

Exemplu:

```
restart_process() {
    # Existing code...

    case "$strategy" in
        # Existing strategies...

        "new_strategy")
            # Implement new restart strategy
            ;;
    esac
```

```
    # Existing code...
}
```

Adăugarea de Noi Opțiuni de Configurare

Pentru a adăuga noi opțiuni de configurare:

1. Adăugați opțiunea în secțiunea corespunzătoare din config.ini
2. Actualizați funcția `read_config()` pentru a analiza noua opțiune
3. Adăugați validare pentru noua opțiune
4. Actualizați documentația pentru a descrie noua opțiune

Exemplu:

```
# In read_config()
NEW_OPTION=$(sed -n '/^\[section\]/,/^\[/p' "$CONFIG_FILE" | grep
"^new_option[[:space:]]*=" | cut -d'=' -f2 | sed
's/^[[:space:]]*//;s/[[:space:]]*$//')
NEW_OPTION=${NEW_OPTION:-default_value}  # Set default if not specified

# Validate
if ! [[ "$NEW_OPTION" =~ ^[0-9]+$ ]]; then
    log "ERROR" "Invalid value for new_option: $NEW_OPTION"
    exit 1
fi

# Export for use in script
export NEW_OPTION
```

Adăugarea de Câmpuri în Baza de Date

Pentru a adăuga noi câmpuri în baza de date:

1. Actualizați scriptul setup.sql pentru a include noile câmpuri
2. Actualizați funcția `get_alarm_processes()` pentru a include noile câmpuri
3. Actualizați orice alte funcții de bază de date care trebuie să utilizeze noile câmpuri
4. Actualizați documentația pentru a descrie noile câmpuri

Exemplu:

```
# In get_alarm_processes()
mysql --defaults-file="$MYSQL_TEMP_CONFIG" -N <<EOF
SELECT CONCAT(p.process_id, '|', p.process_name, '|', s.alarma, '|',
s.sound, '|', s.new_field, '|', s.notes)
FROM STATUS_PROCESS s
JOIN PROCESE p ON s.process_id = p.process_id
WHERE s.alarma = 1 AND s.sound = 0;
EOF
```

Ghid de Testare

Testare Unitară

Pentru testarea funcțiilor individuale:

1. Creați un script de test care importă scriptul principal, dar nu rulează bucla principală
2. Suprascrieți funcțiile de bază de date pentru a utiliza date de test
3. Apelați funcția de testat cu diverse intrări
4. Verificați dacă rezultatele corespund celor așteptate

Exemplu:

```
#!/bin/bash
# Source the main script but don't run main
source ./monitor_service.sh

# Override database functions
get_alarm_processes() {
    echo "1|test_process|1|0|Test notes"
}

# Test restart_process function
restart_process "test_process"
# Verify results
```

Testare de Integrare

Pentru testarea întregului serviciu:

1. Utilizați scriptul test_alarm.sh pentru a seta procesele în stare de alarmă
2. Rulați scriptul monitor_service.sh
3. Verificați dacă procesele sunt repornite și starea de alarmă este ștearsă
4. Verificați jurnalele pentru mesajele așteptate

Exemplu:

```
# Set a process in alarm state
./test_alarm.sh  # Select option 2, then process ID 1

# In another terminal, run the service
sudo ./monitor_service.sh

# Verify the process was restarted and alarm cleared
mysql -u root -p -e "USE v_process_monitor; SELECT alarma FROM
STATUS_PROCESS WHERE process_id = 1;"
# Should return 0
```

Considerații de Performanță

Interogări Bază de Date

Serviciul minimizează încărcarea bazei de date:

1. Interogările rulează doar la intervalul de verificare configurat
2. Interogări SQL eficiente cu operațiuni JOIN
3. Sunt recuperate doar procesele în stare de alarmă

Gestionarea Proceselor

Serviciul este proiectat pentru o gestionare eficientă a proceselor:

1. Utilizează systemd când este posibil pentru o gestionare fiabilă a serviciilor
2. Implementează verificări de sănătate pentru a confirma repornirile reușite
3. Utilizează circuit breaker pentru a preveni încercările excesive de repornire

Utilizarea Memoriei

Serviciul are cerințe minime de memorie:

1. Utilizează array-uri asociative pentru starea circuit breaker-ului
2. Curăță fișierele temporare
3. Implementează rotația jurnalelor pentru a preveni problemele de spațiu pe disc

Depanare

Debugging

Pentru a depana serviciul:

1. Rulați scriptul cu debugging bash:

```
bash -x ./monitor_service.sh
```

2. Adăugați jurnalizare de debug suplimentară:

```
log "DEBUG" "Variable value: $variable"
```

3. Verificați starea bazei de date:

```
mysql -u root -p -e "USE v_process_monitor; SELECT * FROM
STATUS_PROCESS JOIN PROCESE USING (process_id);"
```