Process Monitor Service

Overview

The Process Monitor Service is a robust system monitoring solution designed for RedHat Linux environments. It continuously monitors critical system processes by checking their status in a MySQL database and automatically restarts them when they enter an alarm state. The service implements advanced features such as circuit breaker patterns to prevent cascading failures, configurable health checks to verify successful restarts, and comprehensive logging with automatic rotation.

Table of Contents

System Architecture

The Process Monitor Service consists of the following components:

Core Components

1. **Monitor Service Script** (`monitor_service.sh`)
    - Main service script that implements the monitoring and restart logic
    - Reads configuration from config.ini
    - Queries the database for processes in alarm state
    - Implements restart strategies and circuit breaker patterns
    - Performs health checks after restarts
    - Manages logging with rotation
2. **Configuration File** (`config.ini`)
    - Contains database connection settings
    - Defines monitoring parameters
    - Configures logging behavior
    - Specifies process-specific restart strategies and health checks
3. **Systemd Service Definition** (`monitor_service.service`)
    - Integrates the service with systemd
    - Ensures the service starts automatically at boot
    - Manages service lifecycle (start, stop, restart)

4. **MySQL Database**
    - Stores process definitions and status information
    - Provides the interface for setting processes in alarm state
    - Tracks alarm history through notes field

Workflow

1. The service reads its configuration from `config.ini`
2. It connects to the MySQL database using the configured credentials
3. It queries the database for processes in alarm state (alarma=1)
4. For each process in alarm state:
    - It checks if the circuit breaker is closed for that process
    - If closed, it attempts to restart the process using the configured strategy
    - It performs health checks to verify successful restart
    - It updates the database to clear the alarm state
    - It updates the circuit breaker state based on restart success/failure
5. It waits for the configured interval before checking again

Circuit Breaker Pattern

The service implements a circuit breaker pattern to prevent excessive restart attempts:

1. **Closed State**: Normal operation, restart attempts are allowed
2. **Open State**: After multiple failures, restart attempts are blocked
3. **Reset Mechanism**: Circuit breaker automatically resets after a configured time period

Installation Guide

Prerequisites

- RedHat Linux or compatible distribution
- MySQL Server 5.7 or higher
- Bash 4.0 or higher
- systemd

Installation Steps

1. **Create Service Directory**

```
sudo mkdir -p /opt/monitor_service
```

2. **Copy Service Files**

```
sudo cp monitor_service.sh config.ini /opt/monitor_service/
sudo cp monitor_service.service /etc/systemd/system/
sudo cp monitor_service.8 /usr/share/man/man8/
```

3. **Set Permissions**

```
sudo chmod 755 /opt/monitor_service
sudo chmod 700 /opt/monitor_service/monitor_service.sh
sudo chmod 600 /opt/monitor_service/config.ini
sudo chown -R root:root /opt/monitor_service
```

4. **Set Up Database**

```
# Start MySQL service if not running
sudo systemctl start mysqld

# Create database and tables
sudo mysql < setup.sql
```

5. **Configure the Service**

   Edit the configuration file to set your database credentials and other settings:

```
sudo nano /opt/monitor_service/config.ini
```

6. **Enable and Start the Service**

```
sudo systemctl daemon-reload
sudo systemctl enable monitor_service
sudo systemctl start monitor_service
```

7. **Update Man Pages Database**

```
sudo mandb
```

Configuration

The service is configured through the `config.ini` file, which contains several sections:

Database Configuration

```
[database]
host = localhost
user = root
```

```
password = your_password
database = v_process_monitor
```

Monitoring Parameters

```
[monitor]
; Check interval in seconds (5 minutes)
check_interval = 300
; Maximum number of restart failures before circuit breaker opens
max_restart_failures = 3
; Circuit breaker reset time in seconds (30 minutes)
circuit_reset_time = 1800
```

Logging Configuration

```
[logging]
; Maximum log file size in KB before rotation (default: 5MB)
max_log_size = 5120
; Number of log files to keep (default: 5)
log_files_to_keep = 5
```

Process-Specific Configuration

Each process can have its own configuration section:

```
[process.apache2]
restart_strategy = service
pre_restart_command = /usr/sbin/apachectl configtest
health_check_command = systemctl is-active apache2
health_check_timeout = 10
restart_delay = 5
max_attempts = 2
```

Default Process Configuration

Default settings for processes without specific configuration:

```
[process.default]
restart_strategy = auto
health_check_command = pgrep %s
health_check_timeout = 5
restart_delay = 2
max_attempts = 2
```

Configuration Parameters

Parameter	Description	Default
restart_strategy	How to restart the process: "service", "process", or "auto"	auto
pre_restart_command	Command to run before restart attempt	(none)
health_check_command	Command to verify successful restart	pgrep %s
health_check_timeout	Maximum time in seconds to wait for health check	5
restart_delay	Delay in seconds between restart attempts	2
max_attempts	Maximum number of restart attempts per cycle	2

Database Schema

The service uses two main tables in the MySQL database:

PROCESE Table

Stores process definitions:

```
CREATE TABLE PROCESE (
    process_id INT PRIMARY KEY,
    process_name VARCHAR(100) NOT NULL
);
```

STATUS_PROCESS Table

Stores process status information:

```
CREATE TABLE STATUS_PROCESS (
    process_id INT PRIMARY KEY,
    alarma TINYINT NOT NULL DEFAULT 0,
    sound TINYINT NOT NULL DEFAULT 0,
    notes VARCHAR(100),
    FOREIGN KEY (process_id) REFERENCES PROCESE(process_id)
);
```

Field Descriptions

Table	Field	Description
PROCESE	process_id	Unique identifier for the process
PROCESE	process_name	Name of the process or service
STATUS_PROCESS	process_id	Foreign key referencing PROCESE.process_id

Table	Field	Description
STATUS_PROCESS	alarma	Flag indicating alarm state (1=alarm, 0=normal)
STATUS_PROCESS	sound	Flag for sound notifications (not used by the service)
STATUS_PROCESS	notes	Text field for additional information

Usage

Service Management

- **Start the Service**

```
sudo systemctl start monitor_service
```

- **Check Service Status**

```
sudo systemctl status monitor_service
```

- **Stop the Service**

```
sudo systemctl stop monitor_service
```

- **Restart the Service**

```
sudo systemctl restart monitor_service
```

- **View Service Logs**

```
sudo journalctl -u monitor_service -f
```

or

```
sudo tail -f /var/log/monitor_service.log
```

Manual Page

The service includes a manual page that can be accessed using:

```
man monitor_service
```

Testing

The service includes a testing script (`test_alarm.sh`) that allows you to simulate alarm conditions:

Running the Test Script

```
chmod +x test_alarm.sh
./test_alarm.sh
```

Test Script Options

The test script provides an interactive menu with the following options:

1. **Display all services** - Shows all monitored services and their current status
2. **Set a specific service in alarm state** - Triggers an alarm for a selected service
3. **Set all services in alarm state** - Triggers alarms for all services
4. **Reset all alarms** - Clears all alarm states

Testing Workflow

1. Use the test script to set a service in alarm state
2. Verify that the monitor service detects the alarm and attempts to restart the service
3. Check the logs to confirm the restart attempt and its outcome
4. Verify that the alarm state is cleared in the database after successful restart

Troubleshooting

Common Issues and Solutions

Service Won't Start

- **Check Log Files**

  ```
  sudo journalctl -u monitor_service
  sudo cat /var/log/monitor_service.log
  ```

- **Verify File Permissions**

  ```
  sudo ls -la /opt/monitor_service/
  ```

- **Check Configuration**

```
sudo cat /opt/monitor_service/config.ini
```

Database Connection Issues

- **Verify MySQL Service**

  ```
  sudo systemctl status mysqld
  ```

- **Check Database Credentials**

  Ensure the credentials in config.ini are correct.

- **Test Database Connection**

  ```
  mysql -u root -p -e "USE v_process_monitor; SELECT * FROM PROCESE;"
  ```

Process Restart Failures

- **Check Process Status**

  ```
  systemctl status <process_name>
  ```

- **Verify Health Check Command**

  Run the health check command manually to see if it works:

  ```
  <health_check_command>
  ```

- **Check Pre-restart Command**

  If configured, run the pre-restart command manually to verify it works.

Log File Analysis

The log file (/var/log/monitor_service.log) contains detailed information about the service's operation:

- **INFO** level messages indicate normal operation
- **WARNING** level messages indicate potential issues
- **ERROR** level messages indicate failures that require attention
- **DEBUG** level messages provide detailed information for troubleshooting

Advanced Features

Circuit Breaker Pattern

The service implements a circuit breaker pattern to prevent excessive restart attempts:

- After a configurable number of restart failures, the circuit breaker "opens"
- While open, no further restart attempts are made for the affected process
- After a configurable reset time, the circuit breaker automatically "closes"
- This prevents cascading failures and resource exhaustion

Health Checks

The service performs health checks after restart attempts to verify success:

- Each process can have a custom health check command
- The health check is considered successful if the command returns exit code 0
- The service retries the health check for a configurable timeout period
- This ensures that processes are truly operational after restart

Log Rotation

The service implements automatic log rotation:

- Logs are rotated when they reach a configurable size limit
- A configurable number of old log files are kept
- This prevents disk space issues from large log files

Restart Strategies

The service supports multiple restart strategies:

- **service**: Uses systemd to restart the process as a service
- **process**: Directly kills and restarts the process
- **auto**: Tries service restart first, then falls back to process restart

This flexibility allows the service to handle both systemd services and standalone processes.