System Health Monitoring Tool Developer Documentation

Ömer Faruk Şener

January 22, 2025

Contents

1 Overview

The **System Health Monitoring Tool** is a Python-based application that periodically collects system metrics (e.g., CPU, RAM usage) and stores them in a local SQLite database. When usage exceeds specified thresholds (like CPU > 80%), it triggers an email alert to notify relevant stakeholders.

1.1 Primary Objectives

- **Continuous Monitoring**: Collect CPU/RAM usage data at regular intervals.
- **Data Persistence**: Maintain a historical record of system metrics using SQLite.
- **Alert Mechanism**: Send automated emails when usage thresholds are exceeded.
- **Extensibility**: The code is structured to allow easy addition of new metrics (e.g., disk usage, network I/O) or advanced features (e.g., dashboards, anomaly detection).

2 Version History

Version	Release Date	Changes / Notes
1.0	YYYY-MM-DD	Initial release. - Collects CPU & RAM usage. - Stores data in SQLite. - Sends email alert for high CPU usage.
1.1	Planned	- Add disk usage monitoring. - Configurable thresholds for CPU, RAM, and disk. - Basic logging improvements.
1.2	Planned	- Introduce a Flask-based dashboard. - Graphical views of real-time and historical metrics.
2.0	Under Consideration	- Agent-based architecture for multi-machine monitoring. - Integration with time-series DB (InfluxDB/TimescaleDB). - ML-based anomaly detection.

3 Core Features

- **CPU and RAM Monitoring**: Continuously retrieves CPU and RAM usage percentages using the `psutil` library.

- **SQLite Database Storage**: Saves each measurement (CPU, RAM, timestamp) in a local SQLite database for historic tracking.
- **Email Notifications**: Sends an email alert (via SMTP) if CPU usage exceeds a configured threshold (default: 80%).
- **Cross-Platform Compatibility**: Designed to run on Windows, macOS, and Linux.

4 System Requirements

1. **Python 3.6+**: Tested primarily on Python 3.8 and 3.9.
2. **Operating System**: Compatible with Windows, Linux, and macOS.
3. **Dependencies**:
   - `psutil` (install via `pip install psutil`)
   - `sqlite3` (built-in)
   - `smtplib`, `email.mime` (built-in)
4. **SMTP Access**: A Gmail account or other SMTP service. For Gmail, an App Password may be needed if 2FA is enabled.

5 Installation

```
# Clone or download the repository
git clone https://github.com/example/system-health-monitor.git

cd system-health-monitor

# Install required dependencies
pip install psutil
# sqlite3, smtplib, email.mime are part of the standard library
```

6 Configuration

All configuration parameters for Version 1.0 are stored as constants in `monitor.py`:

- `CPU_THRESHOLD`: CPU usage percentage that triggers an email alert (default 80.0).
- `CHECK_INTERVAL`: Interval (in seconds) between metric collections (default 10).
- `SENDER_EMAIL`: Email address used to send alerts.
- `SENDER_PASSWORD`: SMTP or App Password for the sender account.
- `RECEIVER_EMAIL`: Recipient email address.
- `DB_NAME`: Name/path of the SQLite database file (`system_health.db`).

7 Project Architecture

```
system-health-monitor/
   monitor.py       # Main application script
   system_health.db # SQLite database (created at runtime)
   README.md        # Basic usage instructions
```

7.1 Data Flow

1. **Timer**: uses `time.sleep` in a loop.
2. **collect_metrics**: obtains CPU and RAM.
3. **insert_metric**: stores data in SQLite.
4. **Threshold Check**: triggers `send_email_alert` if exceeded.

8 How to Run

8.1 Local Execution

```
python monitor.py
```

The script will create `system_health.db` if it doesn't exist, then start collecting metrics.

8.2 Background Execution (Linux/macOS)

nohup:

```
nohup python3 monitor.py &
```

screen or **tmux**:

```
screen -S HealthMon
python3 monitor.py
# detach with Ctrl+a, d
```

8.3 macOS launchd

Create a `.plist` file under `/Library/LaunchAgents/` to run automatically on startup.

8.4 Linux systemd

Create a service file in `/etc/systemd/system/`.

8.5 Windows

Use `pythonw.exe` or create a scheduled task. For a true Windows Service, consider `pywin32`.

9 Database Schema

The SQLite database `system_health.db` has a single table:

```
CREATE TABLE IF NOT EXISTS metrics (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
    cpu_usage REAL,
    ram_usage REAL
);
```

9.1 Querying the Database

```
sqlite3 system_health.db
sqlite> SELECT * FROM metrics ORDER BY id DESC LIMIT 10;
```

10 Email Alert Flow

1. **Threshold Check**: If `cpu_val > CPU_THRESHOLD`, proceed.
2. **Alert Trigger**: Calls `send_email_alert`.
3. **SMTP Connection**: `smtplib.SMTP_SSL('smtp.gmail.com', 465)`.
4. **Dispatch**: Sends email to `RECEIVER_EMAIL`.

Note: Implement a cooldown mechanism to avoid spamming if usage remains high.

11 Future Plans and Roadmap

11.1 v1.1

- Add disk usage monitoring
- Configurable thresholds via a config file
- Basic logging improvements

11.2 v1.2

- Flask-based dashboard with Chart.js or Plotly
- Real-time & historical data visualization

11.3 v2.0

- Agent-based architecture for multiple machines
- Time-series DB integration (InfluxDB, TimescaleDB)
- Advanced anomaly detection (ARIMA, LSTM, etc.)

12 Contributing

1. Fork the repository
2. Create a feature branch (`git checkout -b feature/my-feature`)
3. Commit changes (`git commit -m "Add feature"`)
4. Push to the branch (`git push origin feature/my-feature`)
5. Open a Pull Request describing changes

Coding Standards:

- Follow PEP 8 for style guidelines
- Use docstrings to document functions and modules
- Provide tests (Pytest or unittest)

13 License

Include your project's license here. For example:

```
MIT License
Copyright (c) 2025 ...
```

14 Appendix

14.1 Testing Email Sending

Set `CPU_THRESHOLD` to a very low value (e.g., 1.0) temporarily to trigger an alert quickly.

14.2 Common Issues

- **SMTP Authentication Error**: Verify credentials, check 2FA/App Password.
- **No Data in DB**: Ensure `insert_metric()` is called and file permissions are fine.
- **High CPU but No Alert**: The spike might be short-lived. Decrease interval or sample more frequently.
