

Projekt Beschrieb

Thursday, November 6, 2025

10:47 AM

Automatisierte System Monitoring Version1

Beschreibung: Ein Skript was CPU, RAM, Netzwerk und Festplatten Auslastungen überwacht und bei hohen Werte automatisch eine Benachrichtigung schickt. Dazu sollte es automatisiert werden. Speziell daran ist ein Echtzeit-Dashboard, der bei der Ausführung des Skripts sichtbar ist.

GitHub Repository Link: [Sweazyyy881/M122](#)

1. Projektstruktur

Thursday, November 6, 2025 10:18 AM

Ich habe einen Ordner erstellt und dann Dateien erstellt für funktionen.

Struktur:

```
├─ monitor.ps1
├─ config.json
├─ logs\
└─ README.md
```

Explorer:

OneDrive > ... LB2-Projekt-SystemMonitoring-Rafael_Santos >					Search LB2-Projekt-Sys
Sort View ...					Detail
Name	Status	Date modified	Type	Size	
logs	✓	11/6/2025 9:18 AM	File folder		
config.json	✓	11/6/2025 9:17 AM	JSON Source File		
monitor.ps1	✓	11/6/2025 9:16 AM	Windows PowerShell...		
README.md	✓	11/6/2025 9:19 AM	Markdown Source File		

2. Haupt Skript (monitor.ps1)

Thursday, November 6, 2025

10:44 AM

1. Parameter & Grundkonfiguration

Dieser Skript teil gibt an mit IntervalSeconds, wie oft der Skript wiederholt werden soll.

```
[CmdletBinding()]
param(
    [int]$IntervalSeconds = 10 # Messintervall in Sekunden
)
```

2. Pfade & Konfiguration

\$PSScriptRoot: Verzeichnis, in dem das Skript liegt.

Config.json: Ist die Konfig Datei, in dem die Grenzwerte und Einstellungen sich befinden.

Logs/system.log: Hier werden alle Ereignisse protokolliert.

```
6
7 # 1) Pfade & Konfiguration
8
9 $Root = $PSScriptRoot
10 $ConfigPath = Join-Path $Root 'config.json'
11 $LogDir = Join-Path $Root 'logs'
12 $LogPath = Join-Path $LogDir 'system.log'
13
14 if (-not (Test-Path $ConfigPath)) {
15     Write-Error "config.json nicht gefunden: $ConfigPath"
16     exit 1
17 }
18 if (-not (Test-Path $LogDir)) {
19     New-Item -Path $LogDir -ItemType Directory | Out-Null
20 }
21
```

Falls die "config.json" Datei nicht erstellt ist und nicht gefunden werden kann, wird das Skript abgebrochen mit einer Fehler Meldung. Falls die "logs" Ordner fehlt, wird der Skript nicht abgebrochen, sondern es erstellt automatisch eine "logs" Ordner.

3. Laden der Konfiguration

Liest die config.json Datei und wandelt sie in ein PowerShell-Objekt um.

Erwartete Werte:

- cpu_threshold
- memory_threshold
- disk_threshold

```
LB2-Projekt-SystemMonitoring-Rafael_Santos > .\monitor.ps1
22 $config = Get-Content $ConfigPath -Raw | ConvertFrom-Json
23
24 # Pflichtwerte
25 $cpuLimit = [double]$config.cpu_threshold
26 $memoryLimit = [double]$config.memory_threshold
27 $diskFreeMin = [double]$config.disk_threshold # Mindest-% freier Platz
28
29 # Optionalwerte OHNE '??' (5.1-kompatibel)
30 if ($config.PSObject.Properties['cooldown_minutes'] -and $config.cooldown_minutes -ne $null -and $config.cooldown_minutes -ne '') {
31     $cooldownMinutes = [int]$config.cooldown_minutes
32 } else {
33     $cooldownMinutes = 15
34 }
35
36 if ($config.PSObject.Properties['notify_method'] -and $config.notify_method) {
37     $notifyMethod = $config.notify_method.ToString().ToLower()
38 } else {
39     $notifyMethod = 'dialog'
40 }
41
42 if ($config.PSObject.Properties['notify_timeout_seconds'] -and $config.notify_timeout_seconds -ne $null -and $config.notify_timeout_seconds) {
43     $notifyTimeoutSeconds = [int]$config.notify_timeout_seconds
44 } else {
45     $notifyTimeoutSeconds = 5
46 }
```

4. Logging Funktion

Schreibt Zeit auf und Nachricht in die Logdatei.

```
47
```

```

47
48 # -----
49 # 2) Logging
50 # -----
51 function Write-Log([string]$message) {
52     $ts = Get-Date -Format 'yyyy-MM-dd HH:mm:ss'
53     Add-Content -Path $LogPath -Value "$ts $message"
54 }
55

```

5. Benachrichtigung

Popup wird in einem Hintergrundjob gestartet, damit die Hauptschleife nicht blockiert wird. Falls COM-Objekt nicht funktioniert -> Fallback auf "System.Windows.Forms.MessageBox".

```

55
56 # -----
57 # 3) Benachrichtigung (Dialog)
58 # Nicht-blockierend per Start-Job; Popup schließt nach Timeout
59 # -----
60 function Send-Notification([string]$title, [string]$message, [int]$seconds = 5) {
61     if ($notifyMethod -ne 'dialog') { return }
62
63     Start-Job -ScriptBlock {
64         param($t, $m, $s)
65         try {
66             $ws = New-Object -ComObject WScript.Shell
67             # Popup(Text, TimeoutSek, Titel, TypFlags=48 Exclamation)
68             $null = $ws.Popup($m, $s, $t, 48)
69         } catch {
70             try {
71                 Add-Type -AssemblyName System.Windows.Forms | Out-Null
72                 [System.Windows.Forms.MessageBox]::Show($m, $t, 'OK', 'Warning') | Out-Null
73             } catch {}
74         }
75     } -ArgumentList $title, $message, $seconds | Out-Null
76
77     Write-Log "ALERT: $title - $message"
78 }
79

```

6. Cooldown-Mechanismus

Verhindert, dass bei dauerhaft hoher Last alle paar Sekunden Popups erscheinen. "ShouldAlert" prüft, ob seit der letzten Warnung genug Zeit vergangen ist.

```

79
80 # Cooldown je Metrik
81 $cooldown = New-TimeSpan -Minutes $cooldownMinutes
82 $lastAlertTime = @{
83     CPU = Get-Date '2000-01-01'
84     Memory = Get-Date '2000-01-01'
85     Disk = Get-Date '2000-01-01'
86 }
87 function ShouldAlert([string]$metric) {
88     ((Get-Date) - $lastAlertTime[$metric]) -gt $cooldown
89 }
90

```

7. Systemwerte abrufen

Nutzt die Auslastungs Counter in Prozenten.

```

90
91 # -----
92 # 4) Systemwerte abrufen
93 # -----
94 function Get-SystemStats {
95     try {
96         $cpu = (Get-Counter '\Processor(_Total)\% Processor Time').CounterSamples.CookedValue
97         $memory = (Get-Counter '\Memory\% Committed Bytes In Use').CounterSamples.CookedValue
98         $diskFree = (Get-Counter '\LogicalDisk(_Total)\% Free Space').CounterSamples.CookedValue
99
100         [pscustomobject]@{
101             CPU = [math]::Round($cpu, 2) # Auslastung in %
102             Memory = [math]::Round($memory, 2) # Auslastung in %
103             DiskFree = [math]::Round($diskFree, 2) # Freier Platz in %
104             Time = Get-Date
105         }
106     } catch {
107         Write-Log "ERROR counters: $($_.Exception.Message)"
108         throw
109     }
110 }
111
112

```

8. Hauptschleife

Ist eine Endlosschleife, die holt aktuelle Werte, schreibt sie ins Log, prüft Grenzwerte und Cooldown, zeigt Warnungen und wartet die eingestellte Zeit.

```
113 # -----
114 # 5) Hauptschleife
115 # -----
116 while ($true) {
117     try {
118         $s = Get-SystemStats
119
120         $line = ("{0} CPU: {1}% | RAM: {2}% | Disk frei: {3}%" -f ($s.Time.ToString('HH:mm:ss')), $s.CPU, $s.Memory, $s.DiskFree)
121         Write-Host $line
122         Write-Log ("STATS CPU={0}% RAM={1}% DISKFREE={2}%" -f $s.CPU, $s.Memory, $s.DiskFree)
123
124         if ($s.CPU -gt $cpulimit -and (ShouldAlert 'CPU')) {
125             Send-Notification "Hohe CPU-Auslastung" "CPU: $($s.CPU)% > Grenzwert $cpulimit%" $notifyTimeoutSeconds
126             $lastAlertTime['CPU'] = Get-Date
127         }
128
129         if ($s.Memory -gt $memoryLimit -and (ShouldAlert 'Memory')) {
130             Send-Notification "Hoher RAM-Verbrauch" "RAM: $($s.Memory)% > Grenzwert $memoryLimit%" $notifyTimeoutSeconds
131             $lastAlertTime['Memory'] = Get-Date
132         }
133
134         if ($s.DiskFree -lt $diskFreeMin -and (ShouldAlert 'Disk')) {
135             Send-Notification "Wenig Speicher frei" "Nur $($s.DiskFree)% frei < Mindestwert $diskFreeMin%" $notifyTimeoutSeconds
136             $lastAlertTime['Disk'] = Get-Date
137         }
138     }
139     catch {
140         Write-Warning $_.Exception.Message
141         Write-Log "ERROR loop: $($_.Exception.Message)"
142     }
143
144     Start-Sleep -Seconds $IntervalSeconds
145 }
```

Zusammengefasst

- Config.json steuert alles
- Skript läuft endlos, prüft alle X (eingestellte) Sekunden
- Benachrichtigungen sind nicht blockierend
- Cooldown verhindert Spam
- Logs dokumentieren alles

3. Konfig Datei (config.json)

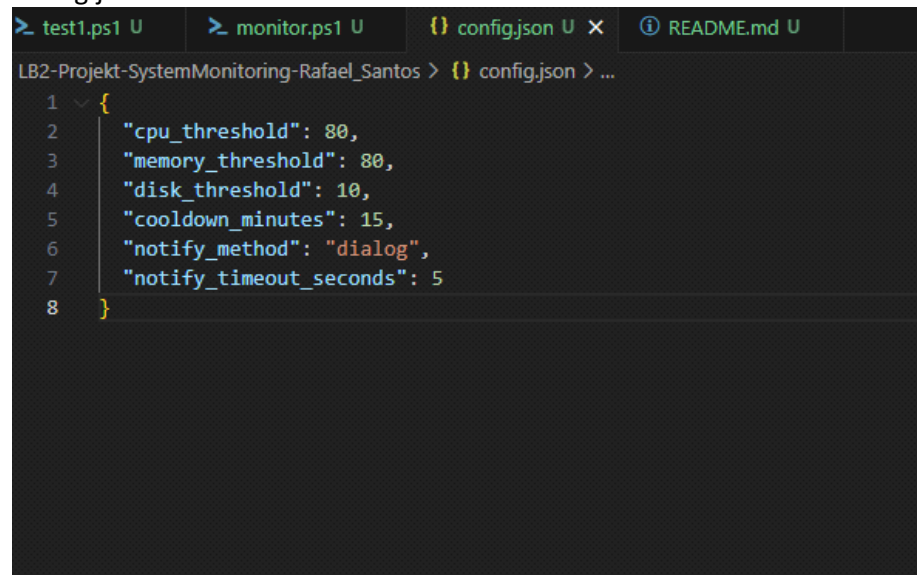
Friday, November 7, 2025 2:42 PM

In dieser Datei kommt Werte rein die man haben möchte mit einer Angabe von dem Prozent Zahl. Der Haupt Skript liest diese Datei beim Start ab und nutzt die Werte für die Angegebenen Angaben.

Vorteil:

- Ich kann die Werte ändern, ohne den Code anzufassen.

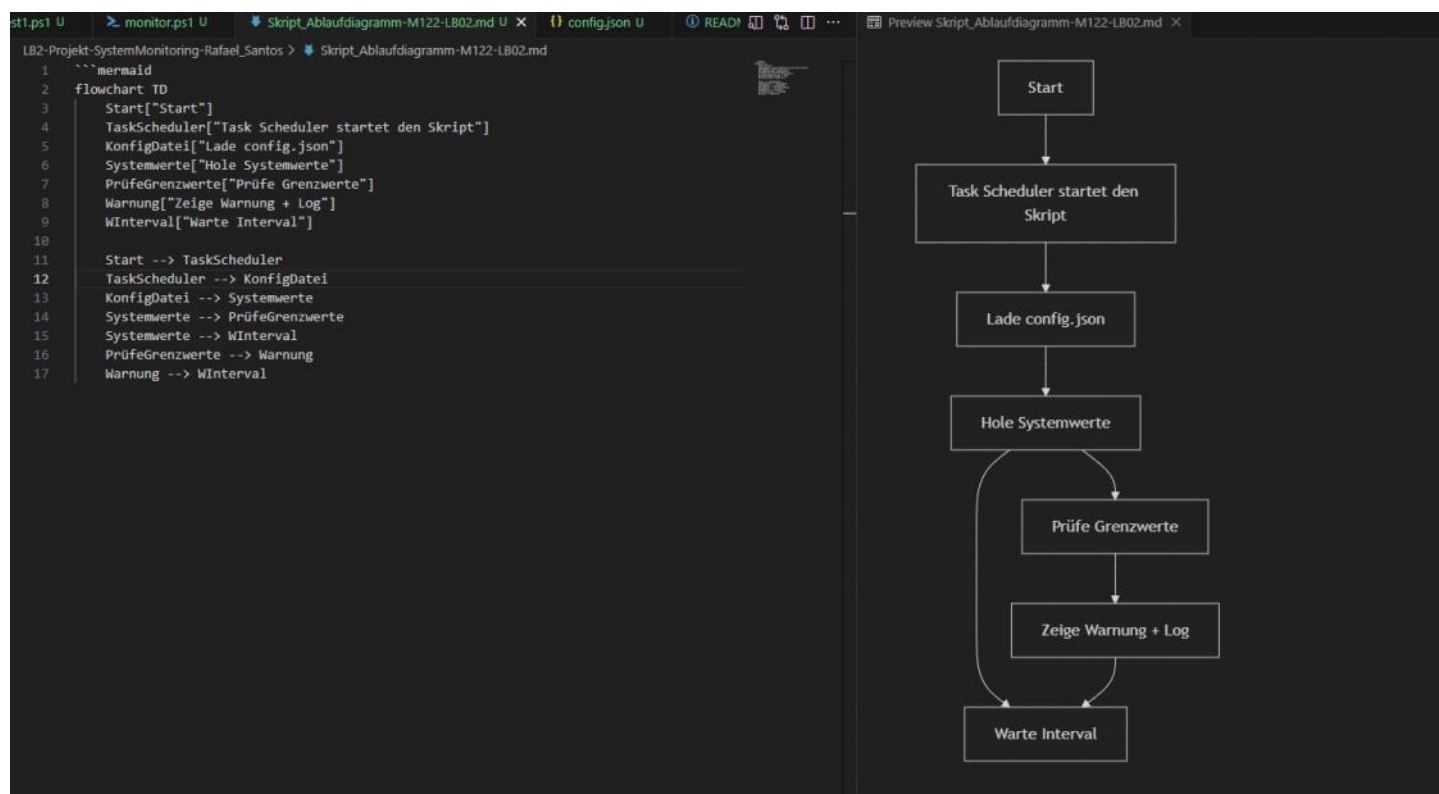
Config.json:



```
1 {  
2   "cpu_threshold": 80,  
3   "memory_threshold": 80,  
4   "disk_threshold": 10,  
5   "cooldown_minutes": 15,  
6   "notify_method": "dialog",  
7   "notify_timeout_seconds": 5  
8 }
```

4. Ablaufdiagramm (Markdown)

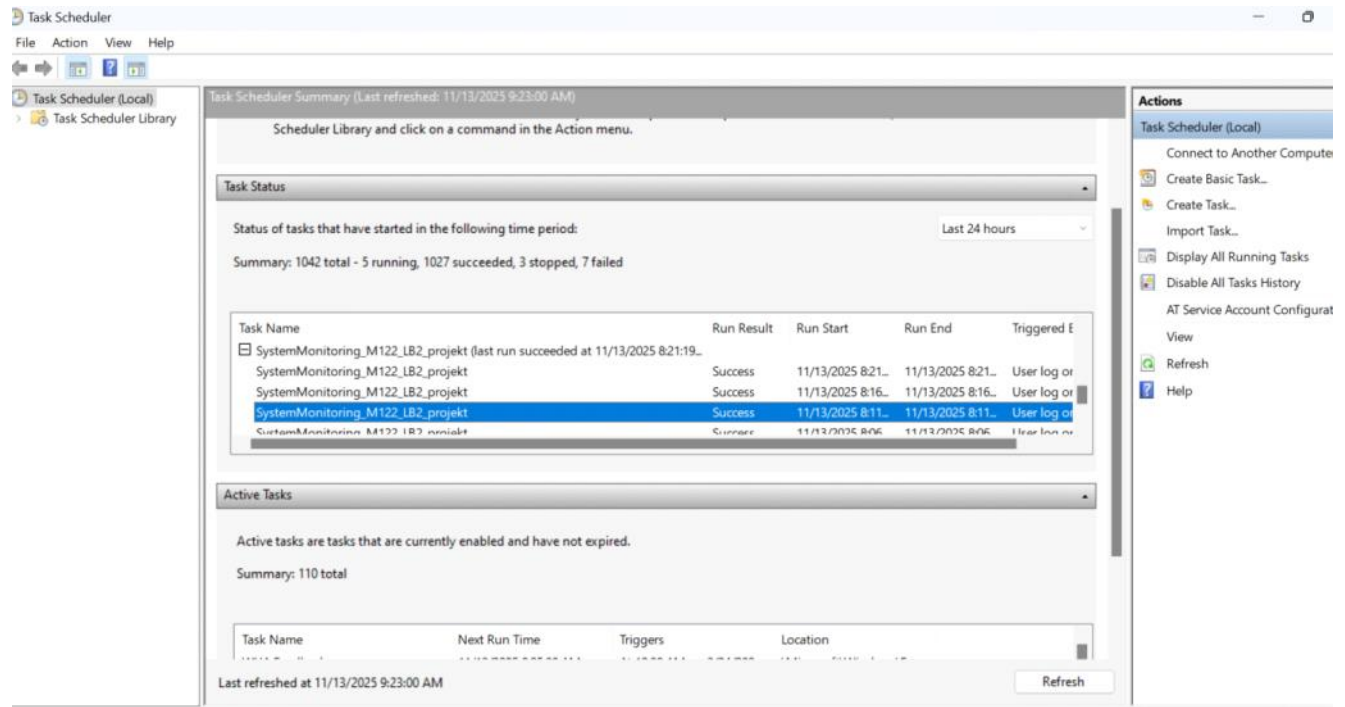
Friday, November 7, 2025 2:42 PM



5. Automatisierung

Thursday, November 13, 2025 9:29 AM

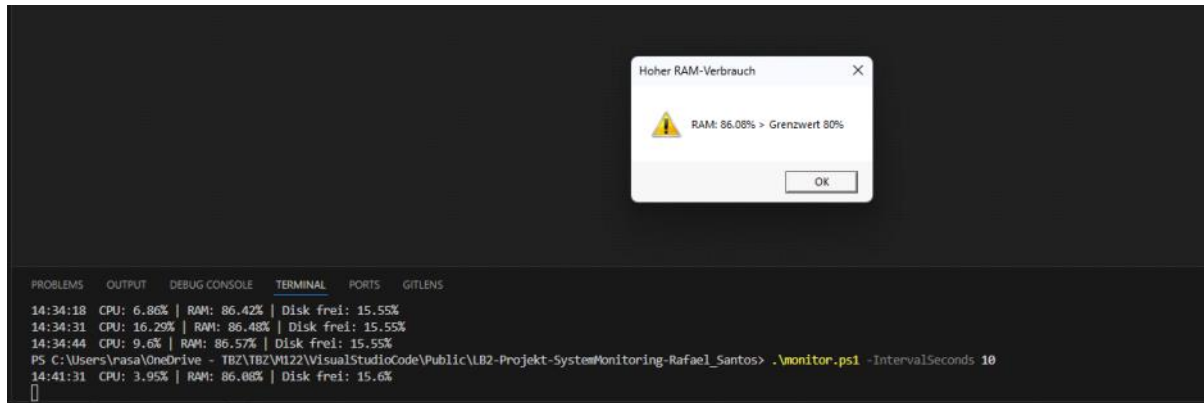
Ich habe für die Automatisierung "Task scheduler" benutzt, damit kann ich eine Task (Aufgabe) erstellen, die in diesem Fall bei jedem Start automatisch den Skript ausführt. Unten im Bild sieht man, dass es beim start meines Laptops gestartet hat und da ich eingestellt habe, dass es für 10min es ausführen soll, hat es nach dieser Zeit beendet.



6. Skript Ausführung

Thursday, November 6, 2025

10:44 AM



Warnung wird angezeigt = Funktioniert

Echtzeit-Dashboard wird angezeigt = Funktioniert