Projekt Beschrieb

Thursday, November 6, 2025 10:47 AM

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

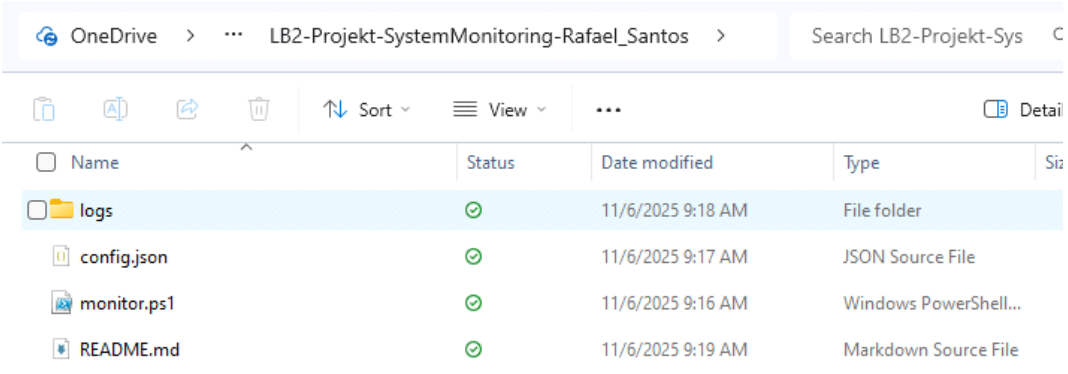
1. Projektstruktur

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:



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.

```powershell
[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.

```powershell
# 1) Pfade & Konfiguration

$Root       = $PSScriptRoot
$ConfigPath = Join-Path $Root 'config.json'
$LogDir     = Join-Path $Root 'logs'
$LogPath    = Join-Path $LogDir 'system.log'

if (-not (Test-Path $ConfigPath)) {
    Write-Error "config.json nicht gefunden: $ConfigPath"
    exit 1
}
if (-not (Test-Path $LogDir)) {
    New-Item -Path $LogDir -ItemType Directory | Out-Null
}
```

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

```powershell
$config = Get-Content $ConfigPath -Raw | ConvertFrom-Json

# Pflichtwerte
$cpuLimit    = [double]$config.cpu_threshold
$memoryLimit = [double]$config.memory_threshold
$diskFreeMin = [double]$config.disk_threshold   # Mindest-% freier Platz

# Optionalwerte OHNE '??' (5.1-kompatibel)
if ($config.PSObject.Properties['cooldown_minutes'] -and $config.cooldown_minutes -ne $null -and $config.cooldown_minutes -ne '') {
    $cooldownMinutes = [int]$config.cooldown_minutes
} else {
    $cooldownMinutes = 15
}

if ($config.PSObject.Properties['notify_method'] -and $config.notify_method) {
    $notifyMethod = $config.notify_method.ToString().ToLower()
} else {
    $notifyMethod = 'dialog'
}

if ($config.PSObject.Properties['notify_timeout_seconds'] -and $config.notify_timeout_seconds -ne $null -and $config.notify_timeout_seconds
    $notifyTimeoutSeconds = [int]$config.notify_timeout_seconds
} else {
    $notifyTimeoutSeconds = 5
}
```

4. Logging Funktion

Schreibt Zeit auf und Nachricht in die Logdatei.

```powershell
47
```

```powershell

    # -------------------------
    # 2) Logging
    # -------------------------
    function Write-Log([string]$message) {
        $ts = Get-Date -Format 'yyyy-MM-dd HH:mm:ss'
        Add-Content -Path $LogPath -Value "$ts $message"
    }

```

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".

```powershell

# -------------------------
# 3) Benachrichtigung (Dialog)
#    Nicht-blockierend per Start-Job; Popup schließt nach Timeout
# -------------------------
function Send-Notification([string]$title, [string]$message, [int]$seconds = 5) {
    if ($notifyMethod -ne 'dialog') { return }

    Start-Job -ScriptBlock {
        param($t, $m, $s)
        try {
            $ws = New-Object -ComObject WScript.Shell
            # Popup(Text, TimeoutSek, Titel, TypFlags=48 Exclamation)
            $null = $ws.Popup($m, $s, $t, 48)
        } catch {
            try {
                Add-Type -AssemblyName System.Windows.Forms | Out-Null
                [System.Windows.Forms.MessageBox]::Show($m, $t, 'OK', 'Warning') | Out-Null
            } catch {}
        }
    } -ArgumentList $title, $message, $seconds | Out-Null

    Write-Log "ALERT: $title - $message"
}

```

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.

```powershell

# Cooldown je Metrik
$cooldown      = New-TimeSpan -Minutes $cooldownMinutes
$lastAlertTime = @{
    CPU    = Get-Date '2000-01-01'
    Memory = Get-Date '2000-01-01'
    Disk   = Get-Date '2000-01-01'
}
function ShouldAlert([string]$metric) {
    ((Get-Date) - $lastAlertTime[$metric]) -gt $cooldown
}

```

7. Systemwerte abrufen

Nutzt die Auslastungs Counter in Prozenten.

```powershell
# -------------------------
# 4) Systemwerte abrufen
# -------------------------
function Get-SystemStats {
    try {
        $cpu      = (Get-Counter '\Processor(_Total)\% Processor Time').CounterSamples.CookedValue
        $memory   = (Get-Counter '\Memory\% Committed Bytes In Use').CounterSamples.CookedValue
        $diskFree = (Get-Counter '\LogicalDisk(_Total)\% Free Space').CounterSamples.CookedValue

        [pscustomobject]@{
            CPU      = [math]::Round($cpu, 2)        # Auslastung in %
            Memory   = [math]::Round($memory, 2)     # Auslastung in %
            DiskFree = [math]::Round($diskFree, 2)   # Freier Platz in %
            Time     = Get-Date
        }
    }
    catch {
        Write-Log "ERROR counters: $($_.Exception.Message)"
        throw
    }
}

```

8. Hauptschleife

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

```powershell
# ------------------------
# 5) Hauptschleife
# ------------------------
while ($true) {
    try {
        $s = Get-SystemStats

        $line = ("{0}  CPU: {1}% | RAM: {2}% | Disk frei: {3}%" -f ($s.Time.ToString('HH:mm:ss')), $s.CPU, $s.Memory, $s.DiskFree)
        Write-Host $line
        Write-Log  ("STATS CPU={0}% RAM={1}% DISKFREE={2}%" -f $s.CPU, $s.Memory, $s.DiskFree)

        if ($s.CPU -gt $cpuLimit -and (ShouldAlert 'CPU')) {
            Send-Notification "Hohe CPU-Auslastung" "CPU: $($s.CPU)% > Grenzwert $cpuLimit%" $notifyTimeoutSeconds
            $lastAlertTime['CPU'] = Get-Date
        }

        if ($s.Memory -gt $memoryLimit -and (ShouldAlert 'Memory')) {
            Send-Notification "Hoher RAM-Verbrauch" "RAM: $($s.Memory)% > Grenzwert $memoryLimit%" $notifyTimeoutSeconds
            $lastAlertTime['Memory'] = Get-Date
        }

        if ($s.DiskFree -lt $diskFreeMin -and (ShouldAlert 'Disk')) {
            Send-Notification "Wenig Speicher frei" "Nur $($s.DiskFree)% frei < Mindestwert $diskFreeMin%" $notifyTimeoutSeconds
            $lastAlertTime['Disk'] = Get-Date
        }
    }
    catch {
        Write-Warning $_.Exception.Message
        Write-Log "ERROR loop: $($_.Exception.Message)"
    }

    Start-Sleep -Seconds $IntervalSeconds
}
```

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)

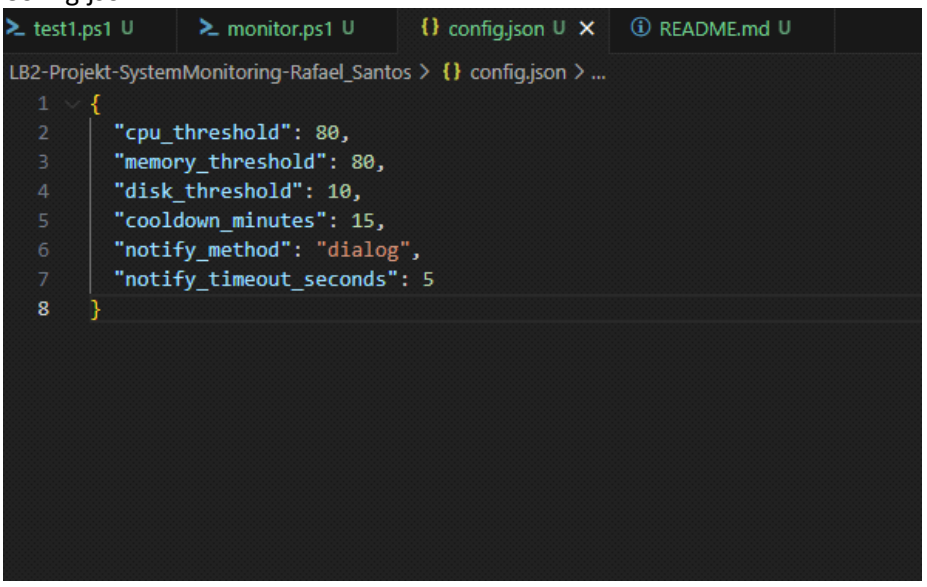
Friday, November 7, 2025 2:42 PM

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

Vorteil:

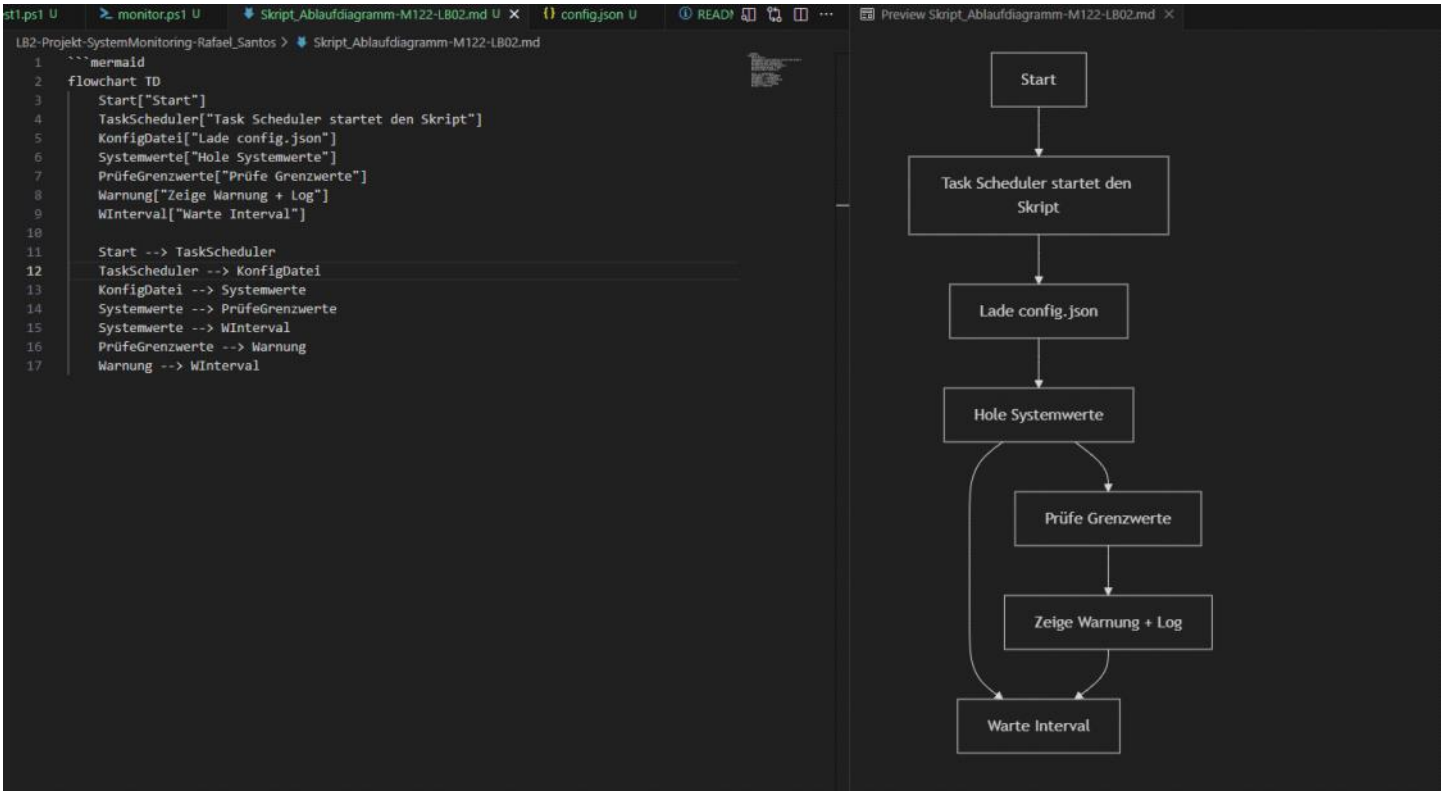
- Ich kann die Werte ändern, ohne den Code anzufassen.

Config.json:



4. Ablaufdiagramm (Markdown)

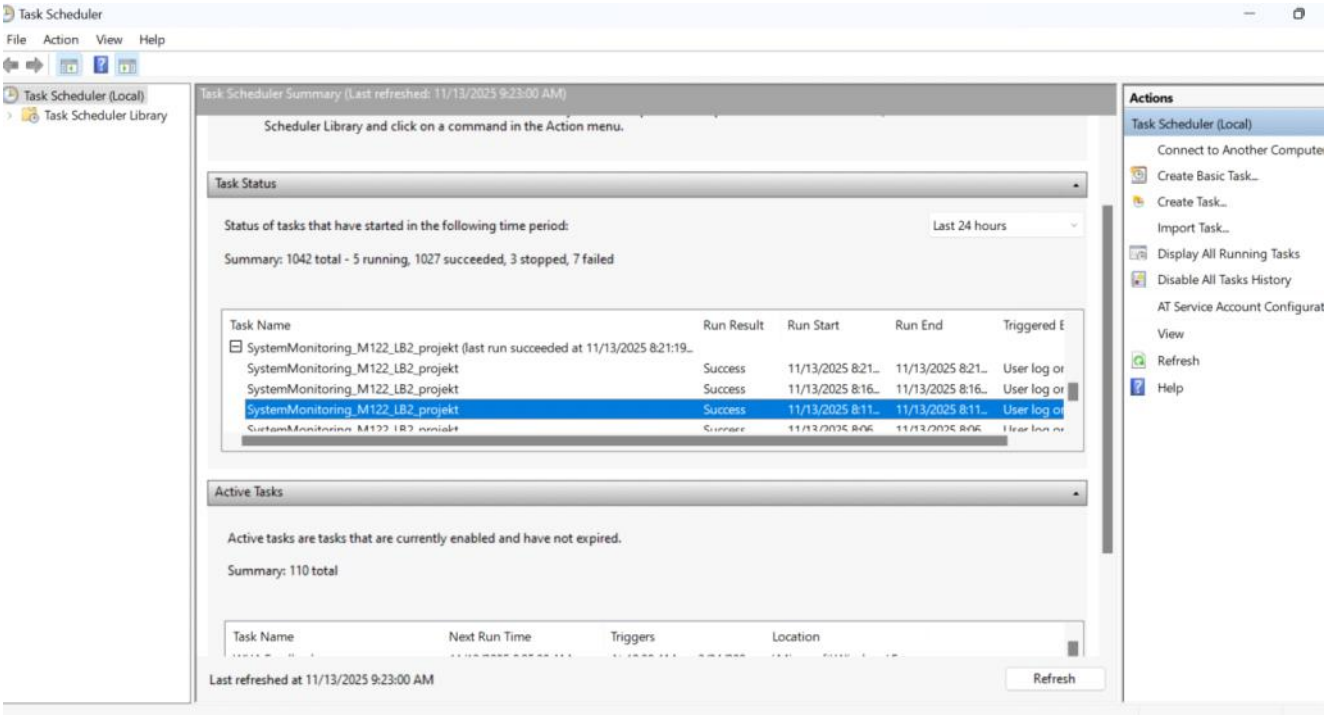
Friday, November 7, 2025 2:42 PM



5. Automatisierung

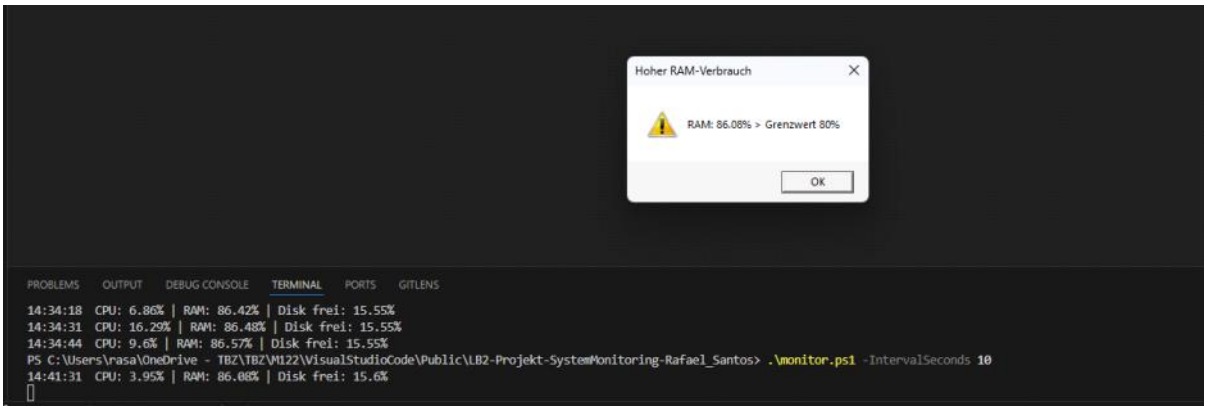
Thursday, November 13, 2025 9:29 AM

Ich habe für die Automatisierung "Task scheduler" benutzt, damit kann ich eine Task (Aufgabe) erstellen, die in diesem Fall bei jeden 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.



5. Skript Ausführung

Thursday, November 6, 2025 10:44 AM



Warnung wird angezeigt = Funktioniert
Echtzeit-Dashboard wird angezeigt = Funktioniert