

**LAPORAN RANCANG BANGUN SYSTEM DAEMON UNTUK
MONITORING LOG SECARA OTOMATIS PADA LINUX**

Dosen Pengampu :
Ferdi Chahyadi, S.Kom., M.Cs



Disusun oleh :
1. Fachrezi Bachry (2401020010)
2. Willy Hadipermana (2401020019)
3. Haikal Fachry Akbar (2401020027)
4. Muhammad Faiz (241020040)

**FAKULTAS TEKNIK DAN TEKNOLOGI KEMARITIMAN
UNIVERTAS MARITIM RAJA ALI HAJI**

KATA PENGANTAR

Puji syukur kami haturkan kehadiran Allah Swt. yang telah melimpahkan rahmat dan hidayah-Nya sehingga kami bisa menyelesaikan karya ilmiah tentang “Rancang Bangun System Daemon Untuk Monitoring Log Secara Otomatis Pada Linux”.

Tidak lupa juga kami mengucapkan terima kasih kepada semua pihak yang telah turut memberikan kontribusi dalam penyusunan karya ilmiah ini. Tentunya, tidak akan bisa maksimal jika tidak mendapat dukungan dari berbagai pihak.

Sebagai penyusun, kami menyadari bahwa masih terdapat kekurangan, baik dari penyusunan maupun tata bahasa penyampaian dalam karya ilmiah ini. Oleh karena itu, kami dengan rendah hati menerima saran dan kritik dari pembaca agar kami dapat memperbaiki karya ilmiah ini.

Kami berharap semoga karya ilmiah yang kami susun ini memberikan manfaat dan juga inspirasi untuk pembaca.

DAFTAR ISI

DAFTAR ISI	ii
A. Abstrak.....	1
B. Pendahuluan	1
C. Rumusan Masalah & Tujuan Project.....	1
D. Perancangan Arsitektur.....	2
E. Implementasi	3
F. Pengeujian & Analisis	4
G. Kesimpulan	5
DAFTAR PUSTAKA.....	6
H. Lampiran(konfigurasi, kode, screenshot)	7
I. Bukti Kerja	16

A. Abstrak

Proyek ini bertujuan untuk merancang dan mengimplementasikan sebuah system daemon berbasis Linux yang berfungsi untuk melakukan pemantauan (monitoring) sistem secara real-time. Daemon ini secara berkala mengumpulkan informasi sistem seperti penggunaan CPU, RAM, disk, status pembaruan paket, riwayat backup, dan log sistem. Data tersebut kemudian diproses dan disajikan dalam bentuk dashboard web interaktif yang dapat diakses melalui browser. Dashboard ini menampilkan informasi dalam format visual yang mudah dipahami, seperti progress bar, tabel, dan notifikasi status. Implementasi menggunakan kombinasi bash scripting, Python, dan JavaScript dengan arsitektur client-server yang ringan. Hasil pengujian menunjukkan bahwa sistem mampu berjalan stabil, memperbarui data setiap 5 detik, dan memberikan informasi yang akurat tentang kondisi sistem.

B. Pendahuluan

Dalam lingkungan sistem operasi Linux, pemantauan sistem adalah aktivitas penting untuk memastikan stabilitas, keamanan, dan kinerja sistem. Administrasi sistem secara manual seringkali memakan waktu dan rentan terhadap human error. Oleh karena itu, diperlukan solusi otomatis yang dapat memantau berbagai aspek sistem secara real-time dan memberikan notifikasi atau laporan visual.

Proyek ini mengembangkan sebuah daemon yang berjalan di latar belakang (background service) untuk mengumpulkan data sistem secara periodik. Data tersebut kemudian diolah dan disajikan melalui antarmuka web yang responsif. Dengan demikian, administrator sistem dapat memantau kesehatan sistem dengan mudah tanpa perlu mengakses server melalui command line.

C. Rumusan Masalah & Tujuan Project

Rumusan Masalah:

1. Bagaimana cara mengotomatisasi pengumpulan data sistem (CPU, RAM, disk, update, backup, log) di Linux?

2. Bagaimana menyajikan data tersebut dalam bentuk visual yang mudah dipahami?
3. Bagaimana memastikan sistem dapat berjalan terus-menerus (daemon) dan diperbarui secara real-time?

Tujuan Project:

1. Membuat daemon yang berjalan di latar belakang untuk monitoring sistem.
2. Mengembangkan dashboard web yang menampilkan data sistem secara real-time.
3. Mengimplementasikan mekanisme pembaruan data otomatis dengan interval tertentu.
4. Menyediakan antarmuka yang intuitif dan informatif bagi administrator sistem.

D. Perancangan Arsitektur

Sistem ini dirancang dengan arsitektur modular yang terdiri dari tiga komponen utama:

1. Daemon / Service Backend
 - Berjalan sebagai systemd service di Linux.
 - Mengeksekusi script bash/Python secara berkala untuk mengumpulkan data.
 - Menyimpan data dalam format JSON (data.json).
2. Web Dashboard (Frontend)
 - Dibangun dengan HTML, CSS, dan JavaScript.
 - Mengambil data dari data.json menggunakan fetch() API.
 - Memperbarui tampilan secara real-time dengan interval 5 detik.
3. Data Flow

Daemon Service → Monitoring Scripts → Output Processing → data.json → Web Dashboard

4. Struktur File:

```
Integrated-System-Monitor-Backup-Daemon-Hybrid-Mode-main/
├── Integrated_daemon.py      # Main Python daemon script
├── README.md                 # Documentation
├── setup_complete.sh         # Installation script
├── config/
│   └── config.json           # Configuration file
└── output/
    ├── integrated-monitor.service # Systemd service file
    ├── backups/                # Backup files
    │   └── backup_*.zip
    └── logs/                   # System logs
        └── updates.log
    └── web/                     # Frontend dashboard
        ├── index.html
        ├── css/
        │   └── style.css
        ├── js/
        │   └── script.js
        └── data.json              # Data output untuk dashboard
    └── laporan/                 # Documentation/report
```

E. Implementasi

1. Daemon Service

- Script bash/Python dijadwalkan menggunakan cron atau systemd timer.
- Contoh tugas yang dijalankan:
 - Mengecek pembaruan paket (pacman -Qu untuk Arch Linux).
 - Memeriksa penggunaan CPU, RAM, disk.
 - Melakukan backup otomatis dan mencatat log.

2. Data Generator

Data sistem dikumpulkan dan disimpan dalam data.json dengan struktur:

```
{  
    "last_check": "timestamp",  
    "resources": { "cpu": 1.2, "ram": {...}, "disk": {...} },  
    "updates": { "count": 19, "list": [...] },  
    "backup": { "current": {...}, "history": [...] },  
    "logs": { "update_logs": [...] }  
}
```

3. Dashboard Web

- HTML (index.html): Struktur halaman dengan section untuk setiap jenis informasi.
- CSS (style.css): Styling untuk kartu, progress bar, tabel, dan layout responsif.
- JavaScript (script.js):
 - Fungsi fetchData() untuk mengambil data JSON.
 - Fungsi updateDashboard() untuk memperbarui UI.
 - Pembaruan otomatis dengan setInterval().

4. Konfigurasi

File config.json berisi pengaturan seperti:

```
{  
    "log_files": ["/var/log/syslog", ...],  
    "keywords": ["error", "warning", ...],  
    "refresh_interval": 5  
}
```

F. Pengeujian & Analisis

1. Pengujian Fungsional

- Daemon: Diuji dengan menjalankan service dan memverifikasi bahwa data JSON diperbarui sesuai interval.
- Dashboard: Diakses melalui browser lokal dan jaringan, responsif di berbagai ukuran layar.
- Data Update: Dipastikan UI memperbarui data setiap 5 detik tanpa perlu refresh manual.

2. Hasil Pengujian

- Sistem berhasil menampilkan:
 - Penggunaan CPU: 1.2%
 - Penggunaan RAM: 4.8 GB / 15.43 GB (31.1%)
 - Pembaruan tersedia: 19 paket
 - Status backup: sukses
 - Log update: 5 file log terbaru

3. Analisis Kinerja

- Interval 5 detik memberikan keseimbangan antara real-time update dan beban sistem.
- Dashboard ringan dan cepat karena hanya menggunakan vanilla JavaScript tanpa framework berat.
- Daemon berjalan dengan konsumsi resource minimal.

G. Kesimpulan

Proyek ini berhasil mengimplementasikan sistem daemon untuk monitoring log dan sistem di Linux, serta dashboard web untuk visualisasi data real-time. Sistem ini dapat membantu administrator dalam memantau kesehatan sistem dengan lebih efisien dan intuitif. Ke depan, sistem dapat dikembangkan dengan fitur notifikasi (email/telegram), autentikasi akses, dan grafik historis.

DAFTAR PUSTAKA

- Khomsah Wiyana, B. I., & Eka, W. S. (2025, Juni). Implementasi Sistem Monitoring Server Berbasis Linux pada ICT Politeknik Negeri Tanah Laut. *Journal Information Technology Trends, II.*
- Saory, F., Jaman, J. H., & Carudin. (n.d.). SISTEM MONITORING SERVER MENGGUNAKAN PROMETHEUS DAN GRAFANA DENGAN INTEGRASI SLACK DI PT. ATLAS LINTAS INDONESIA. *JITET (Jurnal Informatika dan Teknik Elektro Terapan), 13.*
doi:<http://dx.doi.org/10.23960/jitet.v13i3S1.8024>

H. Lampiran(konfigurasi, kode, screenshot)

1. Inisialisasi Daemon (dari integrated_daemon.py, bagian `__init__` dan signal handler)

Potongan kode:

```
class EnhancedDaemon:  
    def __init__(self):  
        self.running = True  
  
        # --- PATH CONFIG ---  
        self.base_dir = '/home/firaz/SO_LATIHAN/log-monitor'  
        self.backup_source = '/home/firaz/Downloads/Data Dummy  
Back Up'  
  
        # Output  
        self.output_dir = os.path.join(self.base_dir, 'output')  
        self.backup_dir = os.path.join(self.output_dir,  
'backups')  
        self.logs_dir = os.path.join(self.output_dir, 'logs')  
        self.json_output = os.path.join(self.base_dir, 'web',  
'data.json')  
        self.update_log = os.path.join(self.logs_dir,  
f'updates_{datetime.now().strftime("%Y%m%d_%H%M%S")}.log')  
  
        # Init Dirs  
        os.makedirs(self.output_dir, exist_ok=True)  
        os.makedirs(self.backup_dir, exist_ok=True)  
        os.makedirs(self.logs_dir, exist_ok=True)  
        os.makedirs(os.path.dirname(self.json_output),  
exist_ok=True)  
  
        # Signal Handler  
        signal.signal(signal.SIGINT, self.stop)  
        signal.signal(signal.SIGTERM, self.stop)  
  
    def stop(self, signum, frame):  
        self.log("🔴 Stopping daemon...")  
        self.running = False
```

Penjelasan kode:

- **Fungsi Utama:** Bagian ini adalah "konstruktor" kelas daemon, yang menginisialisasi semua variabel dan direktori yang dibutuhkan. Ini seperti "persiapan awal" sebelum daemon berjalan: menentukan path folder untuk output (seperti backup dan log), membuat folder jika belum ada, dan menyiapkan handler sinyal (SIGINT dan SIGTERM) agar daemon bisa berhenti dengan aman saat dihentikan (misalnya via Ctrl+C atau perintah systemctl).
- **Mengapa Penting untuk Project:** Ini menunjukkan bagaimana daemon dirancang sebagai layanan otomatis yang stabil di Linux. Tanpa inisialisasi ini, daemon bisa gagal karena path salah atau tidak bisa berhenti properly (misalnya jadi zombie process). Cocok untuk bagian laporan tentang "Desain Sistem Daemon" atau "Inisialisasi Layanan".
- **Cara Kerja yang Mudah Dipahami:** Bayangkan daemon seperti robot penjaga rumah. Di sini, robot pertama-tama cek dan buat "ruangan" (folder) untuk simpan data. Lalu, pasang "alarm" (signal handler) supaya kalau ada perintah mati (dari user atau sistem), robot berhenti dengan rapi tanpa rusak apa-apa. Variabel self.running = True adalah "saklar" utama yang mengontrol loop daemon nanti.

2. Fungsi Monitoring Real-Time (dari integrated_daemon.py, contoh get_cpu_usage dan loop di run)

Potongan kode:

```
def get_cpu_usage(self):  
    try:  
        with open('/proc/stat', 'r') as f:  
            line1 = f.readline()  
            time.sleep(1)  
            with open('/proc/stat', 'r') as f:  
                line2 = f.readline()
```

```

        p1 = [int(x) for x in line1.split()[1:]]
        p2 = [int(x) for x in line2.split()[1:]]

        busy1 = sum(p1[0:3]) + sum(p1[5:8])
        total1 = sum(p1)
        busy2 = sum(p2[0:3]) + sum(p2[5:8])
        total2 = sum(p2)

        delta_total = total2 - total1
        if delta_total == 0: return 0

        delta_busy = busy2 - busy1
        return round((delta_busy / delta_total) * 100, 1)
    except:
        return 0

# ... (potongan lain serupa untuk RAM, disk, uptime)

def run(self):
    self.log("🚀 Enhanced Daemon Started")

    # 1. BOOT TASKS (ONE TIME)
    self.log("📋 Running boot tasks...")
    static_updates = self.check_updates_once()
    static_backup = self.perform_backup_once()
    static_history = self.get_backup_history()
    update_logs = self.get_recent_update_logs()

    boot_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    self.log("✅ Boot tasks completed. Starting real-time
monitor...")

    # 2. REAL-TIME MONITORING LOOP
    while self.running:
        # Real-time metrics
        cpu = self.get_cpu_usage() # This sleeps 1 second
        ram = self.get_memory_usage()
        disk = self.get_disk_usage()
        uptime = self.get_uptime()

        # Combine all data
        data = { ... } # (ringkasan: kumpulkan data ke dict)

```

```

# Write to JSON
try:
    with open(self.json_output, 'w') as f:
        json.dump(data, f, indent=2)
except Exception as e:
    print(f"Error writing JSON: {e}")

```

Penjelasan kode:

- **Fungsi Utama:** get_cpu_usage adalah contoh fungsi monitoring real-time yang membaca file sistem Linux (/proc/stat) untuk hitung persentase CPU usage. Loop di run jalankan tugas boot (sekali saja saat start) lalu loop forever untuk update metrik setiap detik, simpan ke JSON.
- **Mengapa Penting untuk Project:** Ini inti dari "monitoring log/sistem"—meskipun fokus metrik hardware, bisa diextend untuk log (misalnya baca file dari config.json). Tunjukkan daemon sebagai layanan otomatis yang berjalan background. Cocok untuk bagian "Implementasi Monitoring" atau "Loop Daemon".
- **Cara Kerja yang Mudah Dipahami:** Fungsi CPU seperti "mengintip" data sistem dua kali (dengan jeda 1 detik) lalu hitung perubahan untuk dapat persentase. Loop while self.running seperti jam weker yang tiap detik cek kondisi rumah (CPU, RAM, dll.) dan catat ke buku harian (JSON). Kalau sinyal stop datang, loop berhenti.

3. Tugas One-Shot: Update Check dan Backup (dari integrated_daemon.py, contoh check_updates_once dan perform_backup_once)

Potongan kode:

```
def check_updates_once(self):  
    self.log("🔍 Checking updates (One-Time)...")  
    packages = []  
    distro = "Unknown"  
  
    try:  
        # Detect distro  
        is_arch = os.path.exists('/etc/arch-release')  
        is_debian = os.path.exists('/etc/debian_version')  
  
        if is_arch:  
            distro = "Arch Linux"  
            cmd = ['checkupdates'] if  
os.path.exists('/usr/bin/checkupdates') else ['pacman', '-Qu']  
            res = subprocess.run(cmd, capture_output=True,  
text=True)  
            if res.stdout:  
                for line in res.stdout.strip().split('\n'):br/>                    p = line.split()  
                    if len(p) >= 4:  
                        packages.append({  
                            'name': p[0],  
                            'current': p[1],  
                            'new': p[3]  
                        })  
                    # ... (bagian Debian serupa)  
  
                    # Write to log file  
                    self.write_update_log(packages, distro)  
  
    except Exception as e:  
        self.log(f"Update check failed: {e}")  
  
    return {  
        'count': len(packages),  
        'list': packages,  
        'distro': distro,
```

```

        'log_file': os.path.basename(self.update_log)
    }

def perform_backup_once(self):
    self.log("📦 Performing Boot Backup (One-Time)...")
    if not os.path.exists(self.backup_source):
        return {'status': 'error', 'msg': 'Source not found'}

    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    filename = f"backup_{timestamp}.zip"
    filepath = os.path.join(self.backup_dir, filename)

    try:
        file_count = 0
        with zipfile.ZipFile(filepath, 'w', zipfile.ZIP_DEFLATED)
as zipf:
            for root, dirs, files in os.walk(self.backup_source):
                for file in files:
                    p = os.path.join(root, file)
                    zipf.write(p, os.path.relpath(p,
self.backup_source))
                    file_count += 1

        size_mb = round(os.path.getsize(filepath) / (1024*1024),
2)
        self.log(f"✅ Backup finished: {filename}")
        return {
            'status': 'success',
            'filename': filename,
            'size': f"{size_mb} MB",
            'files': file_count,
            'timestamp': timestamp
        }
    except Exception as e:
        return {'status': 'error', 'msg': str(e)}

```

Penjelasan kode:

- **Fungsi Utama:** check_updates_once cek update paket berdasarkan distro (Arch/Debian) menggunakan command-line tools, lalu simpan ke log. perform_backup_once buat ZIP backup dari folder sumber.

- **Mengapa Penting untuk Project:** Ini tunjukkan daemon bukan hanya monitoring pasif, tapi juga tugas aktif seperti backup (relevan untuk "otomatisasi"). Bisa hubungkan dengan monitoring log jika extend untuk scan error di log.
 - **Cara Kerja yang Mudah Dipahami:** Update check seperti "ngecek daftar belanja" via perintah sistem, lalu tulis laporan. Backup seperti "kompres folder" jadi ZIP, hitung ukuran, dan simpan. Dijalankan sekali saat boot untuk efisiensi.
4. Setup Daemon sebagai Service Systemd (dari setup_complete.sh, bagian pembuatan service file)

Potongan kode:

```
# Step 5: Create/update service file
echo "⚙️ Step 5: Creating systemd service file..."
cat > "$SERVICE_FILE" << EOFSERVICE
[Unit]
Description=Integrated System Monitor and Backup (Hybrid Mode)
Documentation=https://github.com/yourrepo/log-monitor
After=network.target multi-user.target
Wants=network.target

[Service]
Type=simple
Restart=always
User=root
Group=root
WorkingDirectory=$BASE_DIR
ExecStart=/usr/bin/python3 $BASE_DIR/integrated_daemon.py

# Timeout untuk eksekusi
TimeoutStartSec=300

# Logging
StandardOutput=journal
StandardError=journal
SyslogIdentifier=$SERVICE_NAME

# Security
Environment=PYTHONUNBUFFERED=1
```

```
[Install]
WantedBy=multi-user.target
EOF
```

Penjelasan kode:

- **Fungsi Utama:** Script ini buat file service systemd untuk jalankan daemon otomatis saat boot.
- **Mengapa Penting untuk Project:** Ini bagian kunci "Rancang Bangun System Daemon"—menjadikan script Python sebagai layanan Linux asli. Cocok untuk "Instalasi dan Konfigurasi Layanan".
- **Cara Kerja yang Mudah Dipahami:** Seperti "daftarkan robot ke sistem" supaya start otomatis. [Unit] tentukan dependensi, [Service] cara jalan (sebagai user root, restart jika crash), [Install] aktifkan saat boot.

5. File Konfigurasi (seluruh config.json)

Potongan kode:

```
{
  "log_files": [
    "/var/log/syslog",
    "/var/log/auth.log",
    "/var/log/kern.log",
    "/var/log/dpkg.log",
    "/var/log/messages",
    "/var/log/secure"
  ],
  "keywords": [
    "failed",
    "error",
    "warning",
    "denied",
    "rejected",
    "invalid",
    "timeout",
    "crash",
    "permission denied",
  ]
}
```

```
        "authentication failure",
        "segmentation fault"
    ],
    "output_file": "output/server.log",
    "json_output": "web/data.json",
    "refresh_interval": 5
}
```

Penjelasan kode:

- **Fungsi Utama:** Config untuk daftar file log yang dimonitor dan keyword error untuk deteksi masalah.
- **Mengapa Penting untuk Project:** Ini tunjukkan fleksibilitas—daemon bisa diubah tanpa ubah kode. Hubungkan dengan "monitoring log" yang kamu sebut.
- **Cara Kerja yang Mudah Dipahami:** Seperti "daftar tugas" untuk daemon: cek file-file ini, cari kata-kata error, simpan output ke file/JSON, refresh tiap 5 detik.

I. Bukti Kerja



