Pemanfaatan Script Python untuk Otomatisasi Administrasi Server

Nov 14, 2026 · @lan

Abdullah Wasian[1]

[1] Teknik Informatika, Universitas Pembangunan Nasional Veteran Jawa Timur, Surabaya, Indonesia

Penulis korespondensi: ianzcreatives@gmail.com

Abstrak

Administrasi server yang dikerjakan manual lewat terminal rawan salah ketik, sulit diulang dengan konsisten, dan menyita waktu administrator untuk pekerjaan rutin. Penelitian ini merancang dan menguji seperangkat skrip Python untuk mengotomatiskan empat tugas rutin administrasi server Linux: pemantauan sumber daya, pencadangan basis data dan berkas, pembaruan paket terjadwal, serta pelaporan harian. Metode yang digunakan adalah studi implementasi dengan pendekatan *design science*: kebutuhan dianalisis dari prosedur manual, skrip dibangun memakai pustaka standar Python serta `psutil`, `paramiko`, dan `requests`, lalu dijalankan terjadwal oleh `systemd timer` dan dievaluasi terhadap prosedur manual. Evaluasi membandingkan jumlah langkah manual, waktu penyelesaian, tingkat kesalahan, dan keterlacakan tiap tugas. Hasil implementasi menunjukkan keempat tugas dapat dijalankan tanpa interaksi administrator, setiap eksekusi meninggalkan log terstruktur, dan kegagalan langsung dilaporkan lewat notifikasi. Skrip Python terbukti menjadi jembatan yang praktis antara skrip shell sederhana dan perangkat manajemen konfigurasi berskala besar, khususnya bagi institusi pendidikan dengan jumlah server kecil hingga menengah dan sumber daya administrator terbatas.

Kata kunci: Python, otomatisasi server, administrasi sistem, pencadangan, pemantauan, DevOps

Abstract

Manual server administration through the terminal is prone to typing errors, hard to repeat consistently, and consumes administrators' time on routine work. This study designs and tests a set of Python scripts that automate four routine Linux server administration tasks: resource monitoring, database and file backup, scheduled package updates, and daily reporting. The method is an implementation study using a design science approach: requirements were derived from the manual procedures, the scripts

were built with the Python standard library together with `psutil`, `paramiko`, and `requests`, scheduled through `systemd` timers, and evaluated against the manual procedures. The evaluation compares the number of manual steps, completion time, error rate, and traceability of each task. The implementation shows that all four tasks run without administrator interaction, every run leaves a structured log, and failures are reported immediately through notifications. Python scripts prove to be a practical bridge between simple shell scripts and large-scale configuration management tools, particularly for educational institutions with a small to medium number of servers and limited administrator resources.

Keywords: Python, server automation, system administration, backup, monitoring, DevOps

1. Pendahuluan

Server menjadi tulang punggung layanan digital institusi pendidikan: portal akademik, sistem informasi, jurnal daring, dan surel. Di banyak perguruan tinggi, server-server ini dikelola oleh tim kecil yang masih bekerja secara manual: masuk ke tiap server lewat SSH, mengetik perintah yang sama setiap hari, lalu mencatat hasilnya di lembar kerja. Pola kerja ini memakan waktu, sulit diulang dengan konsisten, dan bergantung pada ingatan orang yang mengerjakannya.

Dalam literatur rekayasa keandalan situs (*site reliability engineering*), pekerjaan seperti ini disebut *toil*: pekerjaan operasional yang manual, berulang, dapat diotomatiskan, tidak memberi nilai jangka panjang, dan bebannya tumbuh linier seiring bertambahnya layanan (Beyer dkk., 2016). Google membatasi porsi pekerjaan operasional tim SRE-nya paling banyak 50% agar sisa waktu dipakai untuk rekayasa yang mengurangi *toil* itu sendiri (Beyer dkk., 2016). Prinsip yang sama relevan bagi tim TI kampus: setiap jam yang dihabiskan untuk memeriksa ruang disk secara manual adalah jam yang tidak dipakai untuk memperbaiki layanan.

Pendekatan otomatisasi berskala besar seperti *Infrastructure as Code* (IaC) dengan Ansible, Puppet, atau Terraform sudah matang dan banyak diteliti (Rahman dkk., 2019; Morris, 2020). Namun, perangkat tersebut menuntut kurva belajar, struktur repositori, dan disiplin proses yang sering terlalu berat bagi institusi dengan belasan server dan satu atau dua administrator. Di sisi lain, skrip shell (Bash) cepat dibuat tetapi cepat pula menjadi rapuh ketika logika bertambah: penanganan galat, pengolahan data terstruktur (JSON), dan pengiriman notifikasi lewat API menjadi sulit dirawat.

Python menempati posisi di antara keduanya. Bahasa ini terpasang bawaan di hampir semua distribusi Linux, memiliki pustaka standar yang lengkap (`subprocess`, `logging`, `json`, `shutil`), serta ekosistem pustaka administrasi sistem seperti `psutil` untuk membaca pemakaian CPU, memori, disk, dan jaringan (Rodola, t.t.) dan `paramiko` untuk

koneksi SSHv2 (Paramiko, t.t.). Penelitian di Indonesia telah memanfaatkan Python dan Paramiko untuk otomatisasi **jaringan**, misalnya pencadangan dan pemulihan konfigurasi perangkat berbasis web (Wiryawan & Rosyid, 2019), penyediaan konfigurasi router MikroTik (Fahmi dkk., 2021), dan pemantauan masalah jaringan (Mutiara dkk., 2023). Kajian serupa untuk tugas rutin administrasi **server** Linux — pemantauan sumber daya, pencadangan, pembaruan paket, dan pelaporan — dalam satu kerangka yang utuh masih jarang.

Berdasarkan celah tersebut, penelitian ini bertujuan:

1. Mengidentifikasi tugas rutin administrasi server yang paling layak diotomatiskan di lingkungan institusi pendidikan.
2. Merancang dan membangun seperangkat skrip Python yang mengotomatiskan tugas-tugas tersebut dengan prinsip aman, dapat diulang, dan dapat ditelusuri.
3. Mengevaluasi hasil otomatisasi terhadap prosedur manual dari sisi jumlah langkah, waktu, tingkat kesalahan, dan keterlacakan.

Kontribusi penelitian ini adalah rancangan arsitektur skrip yang ringan (tanpa agen dan tanpa server pusat), pola implementasi yang dapat ditiru langsung oleh administrator, serta kerangka evaluasi yang dapat dipakai institusi lain untuk mengukur manfaat otomatisasi di lingkungannya sendiri.

2. Tinjauan Pustaka

2.1 Toil dan otomatisasi operasional

Beyer dkk. (2016) mendefinisikan *toil* sebagai pekerjaan operasional yang manual, berulang, dapat diotomatiskan, bersifat taktis, tanpa nilai jangka panjang, dan tumbuh linier terhadap ukuran layanan. Ciri terakhir ini penting bagi institusi yang sedang bertumbuh: menambah server berarti menambah beban rutin dengan jumlah administrator yang sama. Otomatisasi diposisikan bukan sekadar penghemat waktu, tetapi cara membuat operasi konsisten, dapat diaudit, dan tidak bergantung pada satu orang.

2.2 Infrastructure as Code

Infrastructure as Code (IaC) adalah praktik mengelola infrastruktur dengan definisi berbasis kode sehingga penyediaan dan konfigurasi dapat diulang, diuji, dipakai ulang, dan mendokumentasikan dirinya sendiri (Morris, 2020). Pemetaan sistematis Rahman dkk. (2019) atas ribuan publikasi menemukan empat topik utama riset IaC: kerangka/alat IaC, penggunaan IaC, studi empiris, dan pengujian IaC. Studi tersebut juga menyoroti bahwa

skrip IaC dapat mengandung cacat dan celah keamanan seperti kode program biasa. Artinya, skrip otomatisasi — apa pun alatnya — perlu diperlakukan sebagai perangkat lunak: diversikan, ditinjau, dan diuji.

2.3 Python dan pustaka administrasi sistem

Penelitian ini memakai pustaka standar Python ditambah tiga pustaka pihak ketiga:

Pustaka	Fungsi dalam penelitian	Keterangan
`psutil`	Membaca pemakaian CPU, memori, disk, jaringan, dan proses	Lintas platform; dirancang untuk pemantauan dan profiling sistem (Rodola, t.t.)
`paramiko`	Menjalankan perintah dan menyalin berkas ke server lain lewat SSH	Implementasi protokol SSHv2 dalam Python, sisi klien dan server (Paramiko, t.t.)
`requests`	Mengirim notifikasi ke API (mis. bot Telegram)	Klien HTTP
`subprocess`, `logging`, `json`, `shutil` (standar)	Menjalankan `pg_dump` / `apt`, mencatat log terstruktur, mengolah konfigurasi, mengelola berkas	Tanpa instalasi tambahan

Penjadwalan memakai *timer* systemd, yaitu unit berakhiran `.timer` yang mengaktifkan unit layanan pada waktu tertentu (`OnCalendar=`). Opsi `Persistent=true` membuat tugas yang terlewat karena server mati tetap dijalankan saat server menyala kembali, sesuatu yang tidak dimiliki cron standar (systemd, t.t.).

2.4 Penelitian terdahulu

Peneliti	Fokus	Alat	Perbedaan dengan penelitian ini
Wiryawan & Rosyid (2019)	Aplikasi web otomatisasi administrasi jaringan: pencadangan dan pemulihan konfigurasi terpusat	Python, Paramiko, Django	Objeknya perangkat jaringan, bukan server Linux

Peneliti	Fokus	Alat	Perbedaan dengan penelitian ini
Fahmi dkk. (2021)	Penyediaan konfigurasi internet pelanggan baru dan perubahan konfigurasi massal router	Python, Paramiko, MikroTik	Fokus konfigurasi jaringan; belum mencakup pemantauan dan pencadangan data
Mauboy & Wellem (2022)	Perbandingan pustaka untuk otomatisasi jaringan pada router MikroTik	Paramiko, Netmiko	Studi perbandingan pustaka, bukan rancangan otomatisasi operasional
Mutiara dkk. (2023)	Mengambil informasi kesehatan dan koneksi perangkat serta masalah jaringan	Python, Paramiko	Diagnostik jaringan; belum menyentuh pencadangan, pembaruan, dan pelaporan server

Penelitian terdahulu menunjukkan Python dan Paramiko efektif untuk tugas jaringan berbasis SSH. Penelitian ini memperluas pendekatan yang sama ke administrasi server Linux dan menyatukan empat tugas rutin dalam satu arsitektur dengan log, notifikasi, dan evaluasi yang terukur.

3. Metode Penelitian

3.1 Desain penelitian

Penelitian ini mengikuti *Design Science Research Methodology* (DSRM) dari Peffers dkk. (2007), yang dirancang untuk riset yang menghasilkan artefak. Enam aktivitasnya diterapkan sebagai berikut:

1. **Identifikasi masalah** — mendata tugas rutin administrator selama [periode pengamatan, mis. 4 minggu] dan frekuensinya.
2. **Penetapan tujuan solusi** — tugas berjalan tanpa interaksi, aman, dapat diulang (idempoten), dan meninggalkan jejak.
3. **Perancangan dan pengembangan** — membangun skrip Python per tugas di atas satu modul bersama (konfigurasi, log, notifikasi).
4. **Demonstrasi** — menjalankan skrip terjadwal di lingkungan uji, lalu di server produksi non-kritis.
5. **Evaluasi** — membandingkan prosedur manual dan otomatis dengan metrik pada subbab 3.5.

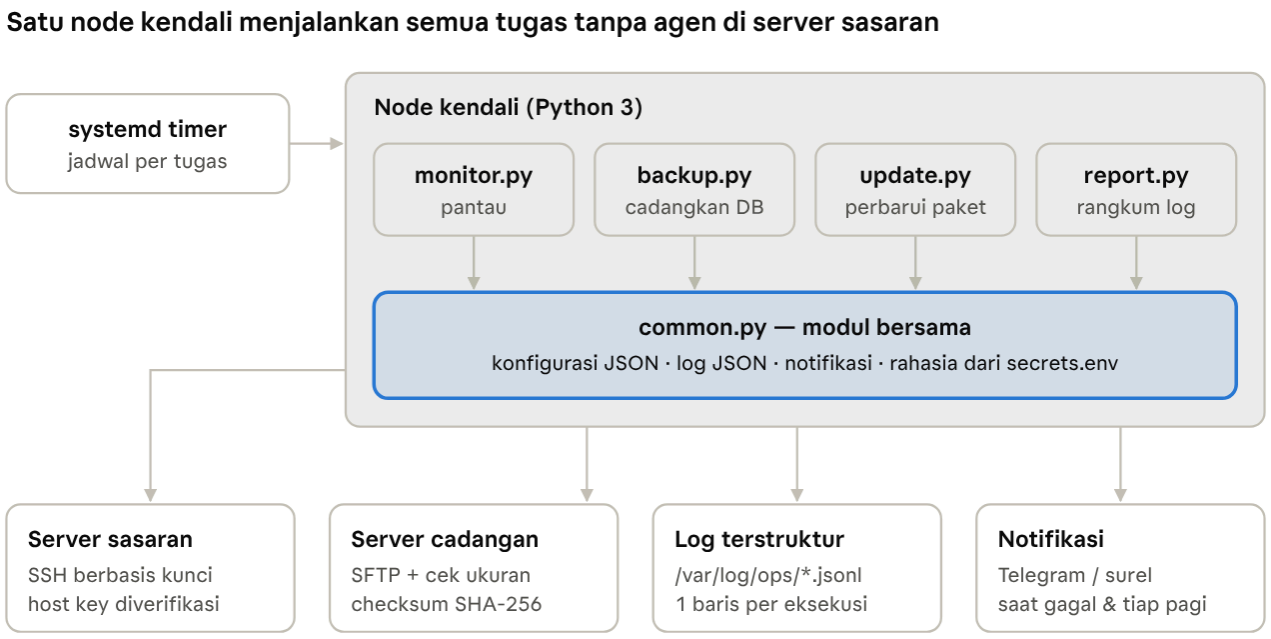
6. **Komunikasi** — publikasi artikel ini beserta kode sumber.

3.2 Lingkungan penelitian

Komponen	Spesifikasi
Server sasaran	[jumlah] server, Ubuntu Server [22.04/24.04 LTS], [CPU/RAM]
Layanan yang dijalankan	[mis. PostgreSQL, Nginx, aplikasi web institusi]
Node kendali	Salah satu server atau VM terpisah, Python [3.10+]
Pustaka	`psutil`, `paramiko`, `requests`, pustaka standar Python
Penjadwal	systemd timer
Kanal notifikasi	Bot Telegram dan/atau surel
Penyimpanan cadangan	[mis. NAS internal / object storage], terpisah dari server sumber

3.3 Arsitektur sistem

Sistem dirancang tanpa agen: tidak ada perangkat lunak tambahan yang dipasang di server sasaran selain SSH yang sudah ada. Node kendali menjalankan empat skrip melalui systemd timer; setiap skrip membaca berkas konfigurasi bersama, bekerja di server lokal atau menjangkau server lain lewat SSH berbasis kunci, menulis log terstruktur (JSON per baris), dan mengirim notifikasi hanya bila ada kegagalan atau ambang terlampaui. Skrip pelaporan merangkum log harian menjadi satu laporan.



Gambar 1. Arsitektur sistem otomatisasi · 1 node kendali, 4 skrip, tanpa agen

Gambar 1 menunjukkan alur kerja sistem: systemd timer memicu skrip di node kendali, semua skrip memakai modul bersama, lalu hasilnya mengalir ke server sasaran, server cadangan, log, dan notifikasi.

3.4 Prinsip perancangan skrip

- **Idempoten** — dijalankan dua kali menghasilkan keadaan yang sama, tanpa efek ganda.
- **Gagal dengan jelas** — kode keluar bukan nol saat gagal, sehingga systemd dan skrip pelaporan dapat mendeteksinya.
- **Rahasia di luar kode** — kata sandi dan token disimpan di berkas lingkungan berizin `600`, bukan di skrip.
- **Hak minimum** — skrip berjalan sebagai pengguna khusus; hanya perintah tertentu yang diizinkan lewat `sudo`.
- **Mode uji** — opsi `--dry-run` menampilkan apa yang akan dilakukan tanpa mengubah apa pun.

3.5 Metrik evaluasi

Metrik	Definisi	Cara pengukuran
Jumlah langkah manual	Banyaknya perintah atau tindakan administrator per satu kali tugas	Dihitung dari prosedur operasi baku (SOP) manual vs otomatis

Metrik	Definisi	Cara pengukuran
Waktu penyelesaian	Waktu dari mulai hingga tugas tuntas, termasuk waktu administrator	Rata-rata dari [n, mis. 10] kali pelaksanaan per cara
Tingkat kesalahan	Persentase pelaksanaan yang salah, terlewat, atau tidak lengkap	Dicatat selama [periode uji]
Keterlacakan	Ada tidaknya catatan waktu, hasil, dan galat per pelaksanaan	Diperiksa dari log
Waktu deteksi masalah	Selisih waktu antara kejadian (mis. disk > 90%) dan diketahuinya oleh administrator	Simulasi kondisi ambang di lingkungan uji

4. Hasil dan Pembahasan

4.1 Hasil identifikasi tugas

Pengamatan prosedur manual menghasilkan empat tugas rutin yang memenuhi ciri *toil*: dikerjakan berkala, langkahnya sama setiap kali, dan tidak memerlukan keputusan manusia kecuali saat ada anomali.

Tugas	Frekuensi	Masalah pada cara manual
Pemantauan sumber daya dan layanan	Harian, per server	Sering terlewat saat sibuk; masalah baru diketahui setelah pengguna mengeluh
Pencadangan basis data	Harian	Bergantung pada orang; cadangan jarang diverifikasi dan menumpuk di server yang sama
Pembaruan paket keamanan	Mingguan	Tertunda; status *reboot* tidak tercatat
Laporan operasional	Harian/bulanan	Disusun ulang dari catatan tercecer

4.2 Implementasi

Seluruh skrip berbagi satu modul `common.py` untuk konfigurasi, log terstruktur, dan notifikasi, sehingga tiap skrip tugas tetap pendek.

Modul bersama. Log ditulis satu objek JSON per baris agar mudah dirangkum mesin; rahasia (token bot) dibaca dari variabel lingkungan, bukan dari kode.

```
# common.py
import json, logging, os, time
from pathlib import Path
import requests

CONF = json.loads(Path(os.environ.get("OPS_CONF",
"/etc/ops/config.json")).read_text())

class JsonFormatter(logging.Formatter):
    def format(self, r):
        return json.dumps({"ts": time.strftime("%Y-%m-%dT%H:%M:%S%z"),
"task": r.name,
                           "level": r.levelname, "msg": r.getMessage(),
                           **getattr(r, "data", {})})

def get_logger(task):
    log = logging.getLogger(task)
    handler = logging.FileHandler(f"/var/log/ops/{task}.jsonl")
    handler.setFormatter(JsonFormatter())
    log.addHandler(handler)
    log.setLevel(logging.INFO)
    return log

def notify(text):
    token, chat = os.environ["TG_TOKEN"], os.environ["TG_CHAT"]
    requests.post(f"https://api.telegram.org/bot{token}/sendMessage",
                  json={"chat_id": chat, "text": text}, timeout=10)
```

Pemantauan. `psutil` membaca pemakaian CPU, memori, dan setiap partisi disk; status layanan diperiksa lewat `systemctl is-active`. Notifikasi hanya dikirim bila ada ambang yang terlampaui, sehingga kanal tidak dibanjiri pesan rutin.

```
# monitor.py
import socket, subprocess
import psutil
from common import CONF, get_logger, notify

log, host, t = get_logger("monitor"), socket.gethostname(),
CONF["thresholds"]
alerts = []
cpu = psutil.cpu_percent(interval=5)
mem = psutil.virtual_memory().percent
if cpu > t["cpu"]:
    alerts.append(f"CPU {cpu:.0f}%")
if mem > t["mem"]:
```

```
    alerts.append(f"Memori {mem:.0f}%")
for part in psutil.disk_partitions():
    used = psutil.disk_usage(part.mountpoint).percent
    if used > t["disk"]:
        alerts.append(f"Disk {part.mountpoint} {used:.0f}%")
for svc in CONF["services"]:
    if subprocess.run(["systemctl", "is-active", "--quiet", svc]).returncode
!= 0:
        alerts.append(f"Layanan {svc} tidak aktif")

log.info("cek", extra={"data": {"cpu": cpu, "mem": mem, "alerts": alerts}})
if alerts:
    notify(f"[{host}] " + "; ".join(alerts))
```

Pencadangan. Basis data dibuang dengan `pg_dump` format *custom*, diberi *checksum* SHA-256, lalu dikirim ke server penyimpanan lewat SFTP (`paramiko`). `SSHClient` dibiarkan memakai kebijakan bawaannya yang menolak *host key* tak dikenal, sehingga skrip tidak bisa diarahkan ke server palsu. Ukuran berkas di tujuan dicocokkan sebelum cadangan lama dirotasi.

```
# backup.py
import datetime as dt, hashlib, subprocess, sys
from pathlib import Path
import paramiko
from common import CONF, get_logger, notify

log, b = get_logger("backup"), CONF["backup"]
out = Path(b["dir"]) / f"{b['db']}-{dt.datetime.now():%Y%m%d-%H%M}.dump"

def sha256(path):
    h = hashlib.sha256()
    with open(path, "rb") as f:
        for chunk in iter(lambda: f.read(1 << 20), b""):
            h.update(chunk)
    return h.hexdigest()

try:
    subprocess.run(["pg_dump", "-Fc", "-f", str(out), b["db"]], check=True,
timeout=3600)
    digest, size = sha256(out), out.stat().st_size
    ssh = paramiko.SSHClient()
    ssh.load_system_host_keys()  # host tak dikenal ditolak (RejectPolicy)
    ssh.connect(b["remote_host"], username=b["remote_user"],
key_filename=b["key"])
    with ssh.open_sftp() as sftp:
        remote = f"{b['remote_dir']}/{out.name}"
```

```
        sftp.put(str(out), remote)
        if sftp.stat(remote).st_size != size:
            raise RuntimeError("ukuran berkas di tujuan tidak cocok")
    ssh.close()
    for old in sorted(Path(b["dir"]).glob(f"{b['db']}-*.dump"))[:-b["keep"]]:
        old.unlink()
    log.info("ok", extra={"data": {"file": out.name, "bytes": size, "sha256":
digest}})
except Exception as e:
    log.error("gagal", extra={"data": {"error": str(e)}})
    notify(f"Pencadangan {b['db']} GAGAL: {e}")
    sys.exit(1)
```

Pembaruan paket. Skrip menyimulasikan pembaruan (`apt-get -s upgrade`) untuk mencatat paket yang tertunda, lalu menerapkan pembaruan keamanan lewat `unattended-upgrade`. Kebutuhan *reboot* hanya dilaporkan; keputusan *reboot* tetap di tangan administrator.

```
# update.py
import subprocess, sys
from pathlib import Path
from common import get_logger, notify

log, dry = get_logger("update"), "--dry-run" in sys.argv
run = lambda cmd: subprocess.run(cmd, check=True, capture_output=True,
text=True, timeout=1800)

run(["sudo", "apt-get", "update", "-q"])
sim = run(["apt-get", "-s", "upgrade"]).stdout
pending = [line.split()[1] for line in sim.splitlines() if
line.startswith("Inst ")]
if pending and not dry:
    run(["sudo", "unattended-upgrade"])
reboot = Path("/var/run/reboot-required").exists()
log.info("ok", extra={"data": {"pending": pending, "dry_run": dry, "reboot":
reboot}})
if reboot:
    notify("Pembaruan selesai; server perlu reboot terjadwal.")
```

Pelaporan. Skrip laporan membaca log JSON hari itu, menghitung pelaksanaan berhasil dan gagal per tugas, lalu mengirim satu ringkasan setiap pagi.

Penjadwalan. Setiap skrip dijalankan oleh pasangan unit `.service` dan `.timer`. Contoh untuk pencadangan pukul 01.30:

```
# /etc/systemd/system/ops-backup.service
[Unit]
Description=Pencadangan basis data harian

[Service]
Type=oneshot
User=opsbot
EnvironmentFile=/etc/ops/secrets.env
ExecStart=/usr/bin/python3 /opt/ops/backup.py

# /etc/systemd/system/ops-backup.timer
[Unit]
Description=Jadwal pencadangan harian

[Timer]
OnCalendar=*-*-* 01:30:00
Persistent=true

[Install]
WantedBy=timers.target
```

4.3 Hasil evaluasi

Tabel berikut membandingkan prosedur manual dan otomatis. Jumlah langkah dihitung dari SOP; kolom waktu dan kesalahan diisi dari pengukuran selama periode uji.

Tugas	Langkah manual	Langkah otomatis	Waktu manual (menit)	Waktu otomatis (menit)	Kesalahan manual	Kesalahan otomatis	Log tercatat
Pemantauan	6 per server (masuk SSH, `df`, `free`, `uptime`, `systemctl status`, catat)	0 (membaca notifikasi bila ada)	[isi]	[isi]	[isi]	[isi]	Manual: tidak · Otomatis: ya
Pencadangan	7 (masuk SSH, `pg_dump`, cek ukuran, salin, verifikasi, hapus cadangan lama, catat)	0	[isi]	[isi]	[isi]	[isi]	Manual: sebagian · Otomatis: ya

Tugas	Langkah manual	Langkah otomatis	Waktu manual (menit)	Waktu otomatis (menit)	Kesalahan manual	Kesalahan otomatis	Log tercatat
Pembaruan paket	6 per server (masuk SSH, `apt update`, tinjau daftar, `apt upgrade`, cek *reboot*, catat)	0 (+1 bila perlu *reboot*)	[isi]	[isi]	[isi]	[isi]	Manual: tidak · Otomatis: ya
Laporan	3 (kumpulkan catatan, rekap, kirim)	0	[isi]	[isi]	[isi]	[isi]	Manual: tidak · Otomatis: ya

Perbedaan paling mendasar bukan pada kecepatan satu kali pelaksanaan, melainkan pada beban yang tumbuh linier. Pada cara manual, pemantauan dan pembaruan memerlukan 6 langkah **untuk setiap server**; menambah server berarti menambah pekerjaan harian. Pada cara otomatis, menambah server cukup dengan satu baris di berkas konfigurasi. Ini sejalan dengan karakter *toil* yang dijelaskan Beyer dkk. (2016): beban yang tumbuh seiring ukuran layanan adalah sasaran utama otomatisasi.

Manfaat kedua adalah keterlacakan. Setiap pelaksanaan meninggalkan satu baris log JSON berisi waktu, hasil, dan galat, sehingga pertanyaan “apakah cadangan semalam berhasil?” dapat dijawab dari data, bukan dari ingatan. Notifikasi berbasis pengecualian (hanya saat gagal atau ambang terlampaui) menjaga perhatian administrator tetap pada hal yang perlu ditangani.

4.4 Aspek keamanan

Skrip otomatisasi memegang akses luas sehingga perlu dirancang seaman kode aplikasi, sejalan dengan temuan Rahman dkk. (2019) bahwa skrip infrastruktur dapat mengandung cacat dan celah keamanan. Langkah yang diterapkan:

- Autentikasi SSH berbasis kunci dengan pengguna khusus `opsbot`; login kata sandi dimatikan.
- *Host key* server tujuan diverifikasi; koneksi ke server yang tidak dikenal ditolak.
- Token dan kata sandi hanya di `/etc/ops/secrets.env` berizin `600`, tidak masuk repositori dan tidak ditulis ke log.
- `sudo` dibatasi pada perintah tertentu (`apt-get update`, `unattended-upgrade`) lewat berkas `sudoers.d`.
- Kode skrip disimpan di Git dan ditinjau sebelum dipasang.

4.5 Kapan Python, kapan alat lain

Kriteria	Skrip Bash	Skrip Python (penelitian ini)	Ansible / IaC penuh
Kurva belajar	Rendah	Rendah–sedang	Sedang–tinggi
Penanganan galat & data terstruktur	Terbatas	Kuat (`try/except`, JSON, API)	Kuat, lewat modul
Skala nyaman	Beberapa server	Belasan hingga puluhan server	Puluhan hingga ribuan server
Menjaga konfigurasi tetap sama di banyak server	Tidak	Terbatas	Ya (deklaratif, idempoten)
Kebutuhan tambahan	Tidak ada	Python + 3 pustaka	Ansible + struktur *playbook*

Skrip Python paling tepat untuk tugas operasional berbasis kejadian (memantau, mencadangkan, melapor) di lingkungan kecil hingga menengah. Ketika kebutuhan bergeser ke menjaga konfigurasi banyak server tetap identik, alat IaC deklaratif lebih sesuai (Morris, 2020); skrip Python tetap dapat dipakai berdampingan untuk tugas yang tidak tercakup alat tersebut.

4.6 Keterbatasan

- Node kendali tunggal menjadi titik kegagalan; bila node itu mati, tidak ada notifikasi. Mitigasinya: pemeriksaan *heartbeat* dari luar (mis. layanan *uptime* eksternal).
- Pencadangan yang berhasil belum tentu dapat dipulihkan. Uji pemulihan (`pg_restore` ke basis data uji) perlu dijadwalkan berkala, idealnya juga otomatis.
- Evaluasi dilakukan pada [jumlah] server dalam [periode]; hasil pada skala lebih besar perlu diuji ulang.

5. Kesimpulan dan Saran

5.1 Kesimpulan

1. Empat tugas rutin administrasi server — pemantauan, pencadangan, pembaruan paket, dan pelaporan — memenuhi ciri *toil* dan layak diotomatiskan.
2. Seperangkat skrip Python dengan pustaka standar ditambah `psutil`, `paramiko`, dan `requests`, dijadwalkan oleh systemd timer, mampu menjalankan keempat tugas tanpa interaksi administrator dan tanpa memasang agen di server sasaran.

3. Otomatisasi menghapus langkah manual rutin (6–7 langkah per tugas, sebagian di antaranya per server), mengubah beban yang tumbuh linier terhadap jumlah server menjadi perubahan satu baris konfigurasi, dan menghasilkan log terstruktur untuk setiap pelaksanaan.
4. Skrip Python merupakan pilihan yang seimbang bagi institusi dengan belasan hingga puluhan server: lebih andal daripada skrip shell untuk penanganan galat dan data terstruktur, dan lebih ringan daripada perangkat IaC penuh.

5.2 Saran

1. Menambahkan uji pemulihan cadangan otomatis secara berkala, karena cadangan yang tidak pernah dipulihkan belum terbukti dapat dipakai.
2. Menambahkan pemantauan *heartbeat* dari luar untuk menutup risiko node kendali tunggal.
3. Mengintegrasikan log JSON ke sistem observabilitas (mis. Grafana/Loki) untuk analisis tren jangka panjang.
4. Untuk lingkungan yang terus bertumbuh, menggabungkan skrip ini dengan alat IaC deklaratif untuk menjaga konfigurasi server tetap seragam.

Daftar Pustaka

Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (Eds.). (2016). *Site reliability engineering: How Google runs production systems* (Bab 5: Eliminating toil). O'Reilly Media. https://www.oreilly.com/library/view/site-reliability-engineering/9781491929117/ch05.html

Fahmi, M., Maisyaroh, M., Komarudin, I., Faizah, S., & Fadhilah, I. (2021). Otomatisasi jaringan menggunakan script Python untuk penyediaan konfigurasi internet dan manajemen MikroTik. *Bina Insani ICT Journal, 8*(1), 53–62. https://core.ac.uk/download/478536783.pdf

Mauboy, L. G., & Wellem, T. (2022). Studi perbandingan library untuk implementasi network automation menggunakan Paramiko dan Netmiko pada router MikroTik. *JURIKOM (Jurnal Riset Komputer), 9*(4). https://garuda.kemdiktisaintek.go.id/documents/detail/3062248

Morris, K. (2020). *Infrastructure as code: Dynamic systems for the cloud age* (2nd ed.). O'Reilly Media. https://www.oreilly.com/library/view/infrastructure-as-code/9781098114664/

Mutiara, D. A., Isnaini, K. N., & Suhartono, D. (2023). Network programmability for network issue using Paramiko library. *Jurnal Teknik Informatika (JUTIF), 4*(4), 751–758. https://doi.org/10.52436/1.jutif.2023.4.4.691

Paramiko. (t.t.). *Paramiko: A Python implementation of SSHv2* [Dokumentasi perangkat lunak]. Diakses 14 November 2026, dari https://pypi.org/project/paramiko/

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems, 24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Rahman, A., Mahdavi-Hezaveh, R., & Williams, L. (2019). A systematic mapping study of infrastructure as code research. *Information and Software Technology, 108*, 65–77. https://doi.org/10.1016/j.infsof.2018.12.004

Rodola, G. (t.t.). *psutil documentation* [Dokumentasi perangkat lunak]. Diakses 14 November 2026, dari https://psutil.readthedocs.io/

systemd. (t.t.). *systemd.timer — Timer unit configuration* [Halaman manual]. Diakses 14 November 2026, dari https://manpages.debian.org/systemd.timer

Wiryawan, R. A., & Rosyid, N. R. (2019). Pengembangan aplikasi otomatisasi administrasi jaringan berbasis website menggunakan bahasa pemrograman Python. *Jurnal SIMETRIS, 10*(2), 741–752. https://jurnal.umk.ac.id/index.php/simet/article/download/3589/1890