ADMIN:DOCS

Linux Administration #9: Systemüberwachung und Performance-Monitoring

Linux-Performance-Monitoring und Systemüberwachung: CPU- und Speicheranalyse, I/O-Metriken mit iostat/vmstat, Log-Diagnose via journalctl, Netzwerk-Monitoring und automatisierte Alerting-Pipelines.

Kategorie: Linux-Administration **Veröffentlicht:** 12.07.2025 **Zuletzt geprüft:** 02.11.2026

Du hast in Artikel #8 unserer Linux-Administration-Serie bereits gelernt, wie du virtuelle Maschinen mit KVM verwaltest, Docker-Container betreibst und LXC-Container einrichtest. Jetzt fragst du dich vielleicht: „Wie erkenne ich eigentlich, ob mein System oder meine virtuellen Maschinen richtig laufen? Woran merke ich, wenn etwas nicht stimmt?“

Genau hier kommt die **Systemüberwachung** ins Spiel – ein absolut essentieller Bereich der Linux-Administration, den du als angehender Administrator unbedingt beherrschen musst.

> **Wichtiger Hinweis:** Dieser Artikel richtet sich an technikinteressierte Linux-Einsteiger, Lernende mit dem Ziel, die Linux-Administration zu verstehen und umzusetzen sowie Leser, die Wert auf praxisnahe, verständliche und gründlich erklärte Inhalte legen.

Warum ist Monitoring so wichtig?

Stell dir vor, du verwaltest einen Linux-Server in einem Unternehmen. Plötzlich beschweren sich Kollegen, dass die Webseite langsam lädt. Oder noch schlimmer: Das System reagiert gar nicht mehr.

Ohne die richtigen Monitoring-Kenntnisse stehst du vor einem Rätsel:

Quelle: https://admindocs.de/de/linux-administration/linux-administration-systemueberwachung-und-performance-monitoring

- Ist die CPU überlastet?
- Ist der Arbeitsspeicher voll?
- Blockiert ein Prozess die Festplatte?
- Liegt ein Netzwerkproblem vor?

> **Das brauchst du später, um:** als Linux-Administrator professionell zu arbeiten, Probleme zu lösen, bevor sie kritisch werden, und um bei Störungen schnell die richtige Ursache zu finden.

Wie dieser Artikel aufgebaut ist

Wir beginnen mit den **Grundlagen der Systemressourcen** – du lernst, was CPU-Load, Speicherverbrauch und I/O-Metriken wirklich bedeuten. Danach arbeiten wir uns durch die wichtigsten Monitoring-Tools mit praktischen Beispielen und typischen Anwendungsfällen.

Im zweiten Teil des Artikels zeige ich dir, wie du Log-Dateien systematisch analysierst und automatisierte Überwachung einrichtest. Abschließend lösen wir gemeinsam realistische Performance-Probleme und erstellen ein einfaches Monitoring-Dashboard.

Praktischer Fokus: Jeder Abschnitt enthält konkrete Beispiele, die du direkt auf deinem System ausprobieren kannst. Du lernst nicht nur die Theorie, sondern wendest alles praktisch an.

> **Das brauchst du später, um:** in der LPIC-1-Prüfung zu bestehen, im Job als Linux-Administrator zu überzeugen und deine eigenen Systeme professionell zu verwalten.

Systemressourcen verstehen

CPU-Auslastung richtig interpretieren

Die CPU-Auslastung ist wahrscheinlich die erste Metrik, die du als Linux-Administrator überwachen wirst. Aber Vorsicht: Hier passieren die meisten Fehlinterpretationen!

Was bedeuten die Load-Average-Werte wirklich?

Wenn du den Befehl uptime eingibst, siehst du drei mysteriöse Zahlen:

```
uptime
14:23:45 up 2 days,  3:42,  2 users,  load average: 0.15, 0.25, 0.30
```

> ⚠ **Typische Fehlinterpretation:** Viele denken, Load 1.0 bedeutet immer 100 % Auslastung. Das stimmt nur bei einem CPU-Kern!

□□ Praktisches Beispiel:

Du hast einen Quad-Core-Prozessor (4 Kerne). Load 1.0 bedeutet also, dass ein Prozess 100 % auf einem CPU-Kern verwendet. Load 2.0 bedeutet, dass ein Prozess 100 % auf zwei CPU-Kernen verwendet.

Dann bedeutet:

- Load 4.0 = 100 % Auslastung aller Kerne
- Load 2.0 = 50 % Auslastung aller Kerne
- Load 8.0 = 200 % Auslastung = System überlastet!

```
nproc
4

cat /proc/cpuinfo | grep "processor" | wc -l
4
```

> **Das brauchst du später, um:** zu verstehen, wann dein System wirklich überlastet ist. Ein Load von 3.0 ist bei einem 8-Kern-System völlig normal, bei einem Single-Core-System aber kritisch.

Unterschied zwischen User-, System- und Idle-Zeit

Mit dem Befehl top siehst du eine detaillierte Aufschlüsselung der CPU-Zeit:

```
top
...
%Cpu(s):  12.5 us,  3.1 sy,  0.0 ni, 84.2 id,  0.2 wa,  0.0 hi,  0.0 si,  0.0 st
```

Was bedeuten diese Werte?

Abkürzung	Bedeutung	Erklärung
us	User	Zeit für Benutzerprogramme (Firefox, LibreOffice, etc.)
sy	System	Zeit für Kernel-Operationen (Dateisystem, Netzwerk, etc.)
ni	Nice	Zeit für Prozesse mit niedriger Priorität
id	Idle	CPU wartet und hat nichts zu tun
wa	I/O Wait	CPU wartet auf Festplatte oder Netzwerk
hi	Hardware IRQ	Zeit für Hardware-Interrupts
si	Software IRQ	Zeit für Software-Interrupts
st	Steal	Zeit, die andere VMs „gestohlen“ haben (nur bei Virtualisierung)

Praktisches Beispiel:

Dein System ist langsam und du siehst:

```
%Cpu(s):   5.2 us,  2.1 sy,  0.0 ni, 12.5 id, 80.2 wa,  0.0 hi,  0.0 si,  0.0 st
```

Diagnose: Hoher wa-Wert (80.2%) bedeutet, die CPU wartet ständig auf die Festplatte. Das Problem ist nicht die CPU, sondern ein I/O-Engpass!

> **Häufiger Fehler:** Viele schauen nur auf die Gesamt-CPU-Auslastung und übersehen, dass das eigentliche Problem bei der Festplatte liegt.

> **Das brauchst du später, um:** Performance-Probleme richtig zu diagnostizieren. Eine hohe CPU-Auslastung kann völlig verschiedene Ursachen haben.

Praktisches Beispiel:

`cat /proc/loadavg` analysieren

Neben `uptime` kannst du die Load-Average-Werte auch direkt aus dem `/proc`-Dateisystem lesen:

```
cat /proc/loadavg
0.15 0.25 0.30 2/267 12345
```

Was bedeuten diese Zahlen?

- `0.15` = Load der letzten 1 Minute
- `0.25` = Load der letzten 5 Minuten

- `0.30` = Load der letzten 15 Minuten
- `2/267` = 2 laufende Prozesse von 267 Gesamtprozessen
- `12345` = PID des zuletzt gestarteten Prozesses

> □□ **Das brauchst du später, um:** Load-Werte in eigenen Monitoring-Skripten zu verwenden:

```
#!/bin/bash
# Einfaches Load-Monitoring-Skript
LOAD_1MIN=$(cat /proc/loadavg | awk '{print $1}')
CPU_CORES=$(nproc)
LOAD_PERCENT=$(echo "scale=2; $LOAD_1MIN / $CPU_CORES * 100" | bc)

echo "Aktuelle Load: $LOAD_1MIN ($LOAD_PERCENT% von $CPU_CORES Kernen)"

if (( $(echo "$LOAD_1MIN > $CPU_CORES" | bc -l) )); then
    echo "⚠□ WARNUNG: System überlastet!"
fi
```

> ⚠ **Typische Fehlinterpretation:** Viele denken, Load 1.0 bedeutet immer 100% Auslastung. Das stimmt nur bei einem CPU-Kern! Bei 4 Kernen wäre Load 4.0 = 100% Auslastung.

Arbeitsspeicher-Verbrauch entschlüsseln

Hier passiert der häufigste Monitoring-Fehler überhaupt: Linux-Einsteiger denken, ihr System hat keinen freien Speicher mehr, obwohl alles völlig normal ist.

Warum Linux-Speicher immer „voll“ aussieht

Schau dir diese typische Ausgabe von `free -h` an:

```
free -h
               total        used        free      shared  buff/cache   available
Mem:           7.7Gi       2.3Gi       1.2Gi       228Mi       4.2Gi       4.6Gi
Swap:          2.0Gi          0B       2.0Gi
```

Panik-Reaktion eines Einsteigers: „Hilfe! Nur noch 1.2 GB frei von 7.7 GB!“

> ⚠ **Das ist völlig falsch!** Linux nutzt "ungenutzten" Speicher intelligent als Cache für Dateien und Programme. Das macht dein System schneller.

Die richtige Interpretation:

- **total:** Gesamter Arbeitsspeicher (7.7 GB)
- **used:** Aktuell von Programmen belegt (2.1 GB)
- **free:** Komplett ungenutzter Speicher (1.2 GB)
- **buff/cache:** Als Cache verwendeter Speicher (4.4 GB) - wird bei Bedarf sofort freigegeben!
- **available:** Tatsächlich verfügbarer Speicher (5.1 GB) - das ist der wichtige Wert!

> □□ **Merksatz:** Der available Wert zeigt dir, wie viel Speicher wirklich zur Verfügung steht. In unserem Beispiel sind das 5.1 GB - völlig ausreichend!

□□ Praktisches Beispiel:

Du startest ein speicherhungriges Programm.

Linux gibt automatisch Cache-Speicher frei:

```
# Vor dem Start eines großen Programms
free -h
               total        used        free      shared  buff/cache   available
Mem:           7.7Gi       2.1Gi       1.2Gi       234Mi       4.4Gi       5.1Gi

# Nach dem Start
free -h
               total        used        free      shared  buff/cache   available
Mem:           7.7Gi       4.8Gi       0.3Gi       234Mi       2.6Gi       2.4Gi
```

> □□ **Was ist passiert?** Das Programm brauchte 2.7 GB Speicher. Linux hat automatisch 1.8 GB aus dem Cache freigegeben. Alles funktioniert perfekt!

Unterschied zwischen „used“, „free“, „available“ und „cached“

Für tiefere Analysen schaust du in `/proc/meminfo`:

- cat /proc/meminfo
- MemTotal: 8052748 kB
- MemFree: 1234567 kB
- MemAvailable: 5234567 kB
- Buffers: 123456 kB
- Cached: 4567890 kB
- SwapCached: 0 kB
- Active: 3456789 kB
- Inactive: 2345678 kB
- Dirty: 12345 kB
- Writeback: 0 kB
- Slab: 345678 kB

Die wichtigsten Werte erklärt:

Wert	Bedeutung	Wann kritisch?
MemTotal	Gesamter RAM	–
MemFree	Komplett ungenutzter RAM	Unwichtig für Bewertung
MemAvailable	Verfügbarer RAM (inkl. freigebbarer Cache)	< 10% = kritisch
Buffers	Puffer für Dateisystem-Metadaten	–
Cached	Cache für Dateiinhalte	Wird automatisch freigegeben
SwapCached	Swap-Speicher im RAM-Cache	0 = System swappt
Active	Kürzlich verwendeter Speicher	–
Inactive	Länger nicht verwendeter Speicher	–
Dirty	Geänderte Daten, noch nicht auf Festplatte	100MB = I/O-Stau
Writeback	Daten werden gerade geschrieben	0 = I/O läuft
Slab	Kernel-Datenstrukturen	500MB = ungewöhnlich

 Typischer Fehler: Einsteiger addieren Cached zu MemFree und denken, das wäre der verfügbare Speicher. Das stimmt nicht! Nutze immer MemAvailable.

 Das brauchst du später, um: Memory-Leaks zu erkennen, Speicher-Engpässe zu diagnostizieren und zu verstehen, wann dein System wirklich zu wenig RAM hat.

Festplattenspeicher und I/O verstehen

Festplatten-Monitoring ist komplex, weil zwei völlig verschiedene Probleme auftreten können: **Speicherplatz-Mangel und Performance-Probleme.**

Inodes vs. Speicherplatz – beide können „voll“ sein

Ein häufiges Problem, das Einsteiger verwirrt: Du bekommst die Fehlermeldung `No space left on device`, obwohl `df` noch freien Speicherplatz anzeigt.

□□ Praktisches Beispiel:

Schauen wir uns beide Werte an:

```
# Speicherplatz prüfen
df -h /
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        20G   15G  4.2G  79% /

# Inodes prüfen
df -i /
Filesystem      Inodes   IUsed   IFree IUse% Mounted on
/dev/sda1      1310720 1310720       0  100% /
```

Problem erkannt? Speicherplatz ist noch da `4.2 GB frei`, aber die Inodes sind zu 100% belegt!

Was sind Inodes?:

- Jede Datei braucht einen Inode (Index-Node)
- Inodes speichern Metadaten: Berechtigungen, Zeitstempel, Speicherort
- Die Anzahl der Inodes ist beim Formatieren festgelegt
- Viele kleine Dateien = viele Inodes verbraucht

> ⚠ **Typisches Szenario:** Log-Dateien werden rotiert und es entstehen Millionen kleiner Dateien. Speicherplatz ist noch da, aber keine Inodes mehr frei.

Lösung finden:

```
# Verzeichnis mit den meisten Dateien finden
find /var/log -type f | wc -l
1234567

# Alte Log-Dateien aufräumen
find /var/log -name "*.log.*" -mtime +30 -delete
```

> □□ **Das brauchst du später, um:** mysteriöse "Disk full"-Fehler zu verstehen und zu lösen, auch wenn scheinbar noch Platz vorhanden ist.

Was bedeuten die verschiedenen I/O-Metriken?

Mit `iostat` bekommst du detaillierte Informationen über die Festplatten-Performance:

```
iostat -x 1
Device            r/s     w/s     rkB/s     wkB/s   rrqm/s   wrqm/s  %util
sda              12.34   45.67    123.45    456.78     0.12     2.34   85.6
```

Jede Spalte erklärt:

Spalte	Bedeutung	Kritischer Wert
r/s	Read-Operationen pro Sekunde	200 bei HDD
w/s	Write-Operationen pro Sekunde	100 bei HDD
rkB/s	Gelesene KB pro Sekunde	–
wkB/s	Geschriebene KB pro Sekunde	–

Spalte	Bedeutung	Kritischer Wert
rrqm/s	Read-Requests merged	Niedrig = schlecht
wrqm/s	Write-Requests merged	Niedrig = schlecht
%util	Auslastung der Festplatte	80% = Engpass

□□ Praktisches Beispiel:

Dein System ist langsam und `iostat` zeigt:

```
Device            r/s     w/s     rkB/s     wkB/s   %util
sda             234.5    12.3   2345.67    123.45   98.7
```

Diagnose: Sehr hohe Read-Rate (234.5 r/s) und 98.7% Auslastung. Die Festplatte ist der Engpass!

Ursache finden:

```
# Welcher Prozess verursacht die meiste I/O?
iotop -o
```

> ⚠ **Häufiger Fehler:** Nur auf die Transferrate (rkB/s, wkB/s) schauen und die Anzahl der Operationen (r/s, w/s) ignorieren. Viele kleine Zugriffe können eine Festplatte genauso blockieren wie wenige große.

> **Das brauchst du später, um:** I/O-Engpässe zu identifizieren, langsame Anwendungen zu optimieren und zu entscheiden, ob du eine SSD brauchst.

Praktisches Beispiel:

Festplatte voll, aber `df` zeigt noch Platz

```
# Du bekommst diese Fehlermeldung:
touch /tmp/testfile
touch: cannot touch '/tmp/testfile': No space left on device

# Aber df zeigt noch Platz:
df -h /tmp
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        10G   8.5G  1.2G  88% /tmp

# Das Problem: Inodes sind voll!
df -i /tmp
Filesystem      Inodes   IUsed   IFree IUse% Mounted on
/dev/sda1      655360  655360       0  100% /tmp
```

Lösung: Alte, kleine Dateien löschen um Inodes freizugeben:

```
# Verzeichnis mit den meisten Dateien finden
find /tmp -type f | head -1000 | xargs ls -la

# Alte temporäre Dateien löschen
find /tmp -name "*.tmp" -mtime +7 -delete
```

Praktische Regel:

```
# Warnung bei 90% Belegung
df -h | awk '$5 > 90 {print "⚠ WARNUNG: " $1 " ist zu " $5 " voll!"}'

# Kritisch bei 95% Belegung
df -h | awk '$5 > 95 {print "🚨 KRITISCH: " $1 " ist zu " $5 " voll!"}'
```


```
# Warnung bei 90% Belegung
df -h | awk '$5 > 90 {print "⚠ WARNUNG: " $1 " ist zu " $5 " voll!"}'

# Kritisch bei 95% Belegung
df -h | awk '$5 > 95 {print "🚨 KRITISCH: " $1 " ist zu " $5 " voll!"}'
```


> 🎯 **Das brauchst du später, um:** Systemausfälle durch volle Festplatten zu vermeiden und Performance-Probleme zu verhindern.

Wichtige Monitoring-Befehle meistern

`top` ist wahrscheinlich der erste Monitoring-Befehl, den du als Linux-Administrator lernst. Er zeigt dir in Echtzeit, was auf deinem System passiert. Aber die meisten Einsteiger nutzen nur einen Bruchteil seiner Möglichkeiten.

Jeden Wert in der `top`-Ausgabe verstehen

Starte `top` und schauen wir uns die Ausgabe gemeinsam an:

```
top
top - 14:23:45 up 2 days,  3:42,  2 users,  load average: 0.15, 0.25, 0.30
Tasks: 267 total,   1 running, 266 sleeping,   0 stopped,   0 zombie
%Cpu(s):  12.5 us,  3.1 sy,  0.0 ni, 84.2 id,  0.2 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   7876.2 total,   1234.5 free,   2345.6 used,   4296.1 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   5123.4 avail Mem

□PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1234 user      20   0  123456  45678  12345 S  25.3   0.6   1:23.45 firefox
   5678 root      20   0   67890  23456   7890 R  12.1   0.3   0:45.67 python3
   9012 user      20   0   34567  12345   4567 S   5.2   0.2   0:12.34 chrome
```

Kopfzeile Zeile für Zeile erklärt:

Zeile 1 – Systemzeit und Load:

- top - 14:23:45 up 2 days, 3:42, 2 users, load average: 0.15, 0.25, 0.30
- 14:23:45 = Aktuelle Uhrzeit
- up 2 days, 3:42 = System läuft seit 2 Tagen und 3:42 Stunden
- 2 users = Zwei Benutzer sind angemeldet
- load average: 0.15, 0.25, 0.30 = Durchschnittslast der letzten 1, 5, 15 Minuten

Zeile 2 – Prozess-Status:

- Tasks: 267 total, 1 running, 266 sleeping, 0 stopped, 0 zombie
- **267 total** = Insgesamt 267 Prozesse
- 1 running = Ein Prozess läuft gerade (meist top selbst)
- 266 sleeping = 266 Prozesse warten auf Ereignisse
- 0 stopped = Keine gestoppten Prozesse (Ctrl+Z)
- zombie = Keine Zombie-Prozesse (gut!)

> ⚠ Achtung bei Zombie-Prozessen: Wenn hier Werte > 0 stehen, hast du ein Problem. Zombie-Prozesse sind "tote" Prozesse, die nicht ordnungsgemäß beendet wurden.

Zeile 3 – CPU-Auslastung:

```
%Cpu(s):  12.5 us,  3.1 sy,  0.0 ni, 84.2 id,  0.2 wa,  0.0 hi,  0.0 si,  0.0 st
```

Zeile 4+5 – Speicher-Information:

- MiB Mem : 7876.2 total, 1234.5 free, 2345.6 used, 4296.1 buff/cache
- MiB Swap: 2048.0 total, 2048.0 free, 0.0 used. 5123.4 avail Mem

- 7876.2 total = Gesamter RAM in MiB
- 1234.5 free = Komplett freier RAM
- 2345.6 used = Von Programmen belegter RAM
- 4296.1 buff/cache = Als Cache verwendeter RAM
- 5123.4 avail Mem = Wichtigster Wert: Verfügbarer RAM

> □□ **Das brauchst du später, um:** schnell zu erkennen, ob dein System genug Speicher hat. Wenn avail Mem unter 10% des Gesamt-RAMs fällt, wird es kritisch.

Interaktive Befehle in `top` (k, r, M, P, etc.)

`top` ist nicht nur ein passives Anzeige-Tool – du kannst es interaktiv steuern:

Die wichtigsten Tastenkürzel:

Taste	Funktion	Wann verwenden?

M	Nach Speicherverbrauch sortieren	Memory-Leaks finden
P	Nach CPU-Verbrauch sortieren	CPU-Fresser identifizieren
T	Nach Laufzeit sortieren	Langläufer finden
k	Prozess beenden (kill)	Hängende Prozesse stoppen
r	Prozess-Priorität ändern (renice)	Wichtige Prozesse bevorzugen
1	Alle CPU-Kerne einzeln anzeigen	Multi-Core-Auslastung prüfen
c	Vollständige Kommandozeile anzeigen	Parameter von Prozessen sehen
q	`top` beenden	–

Praktische Übung: Speicherfresser identifizieren

Starte `top`:

- top

Drücke **M** um nach Speicherverbrauch zu sortieren:

```
PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
1234 user      20   0 2345678 567890 123456 S   5.3  7.2   12:34.56 firefox
5678 user      20   0 1234567 234567  67890 S   2.1  3.0    5:67.89 chrome
9012 root      20   0  567890 123456  45678 S   0.5  1.6    1:23.45 systemd
```

Drücke **c** um die vollständige Kommandozeile zu sehen:

- COMMAND
- firefox --new-window https://admindocs.de
- chrome --disable-extensions --incognito
- /lib/systemd/systemd --switched-root --system

Was bedeuten die Spalten?

Spalte	Bedeutung	Kritische Werte
PID	Prozess-ID	–
USER	Benutzer, der den Prozess gestartet hat	–
PR	Priorität (niedriger = wichtiger)	< 0 = Realtime
NI	Nice-Wert (-20 bis +19)	-20 = höchste Priorität
VIRT	Virtueller Speicher (kann größer als RAM sein)	–
RES	Resident Memory (tatsächlich im RAM)	**Wichtigster Wert!**
SHR	Shared Memory (mit anderen Prozessen geteilt)	–
S	Status (R=running, S=sleeping, Z=zombie)	Z = Problem!
%CPU	CPU-Verbrauch in Prozent	100% bei Multi-Core
%MEM	RAM-Verbrauch in Prozent	10% = auffällig

> ⚠ **Typischer Fehler:** Viele schauen auf die VIRT-Spalte und erschrecken sich. Firefox zeigt 2.3 GB virtuellen Speicher? Das ist normal! Wichtig ist nur RES - der tatsächlich verwendete RAM.

`htop` – Die benutzerfreundliche Alternative

`htop` ist eine moderne Alternative zu `top` mit besserer Benutzeroberfläche:

```
# htop installieren (falls nicht vorhanden)
$ sudo apt install htop

# Debian/Ubuntu
$ sudo dnf install htop
```

```
# Fedora/RHEL
# htop starten
$ htop
```

Vorteile von htop:

- Farbige Darstellung für bessere Übersicht
- Maus-Unterstützung
- Einfachere Navigation mit Pfeiltasten
- Prozess-Baum-Ansicht `F5`
- Eingebaute Hilfe `F1`

□□ Praktisches Beispiel:

In `htop` einen hängenden Prozess beenden:

- Starte htop
- Navigiere mit Pfeiltasten zum problematischen Prozess
- Drücke `F9` (Kill)
- Wähle das Signal (meist `SIGTERM` oder `SIGKILL`)
- Bestätige mit `Enter`

> ⚠ **Warnung:** Verwende SIGKILL (Signal 9) nur als letztes Mittel. Es beendet Prozesse sofort ohne Aufräumen. Versuche zuerst SIGTERM (Signal 15).

> □□ **Das brauchst du später, um:** hängende Prozesse zu identifizieren und sicher zu beenden, CPU- und Memory-Leaks zu finden und die Systemlast zu überwachen.

`iostat` – Festplatten-Performance analysieren

`iostat` ist dein wichtigstes Tool zur Analyse von Festplatten-Performance. Es gehört zum `sysstat`-Paket, das du möglicherweise nachinstallieren musst.

Installation und Grundlagen

```
# sysstat installieren (enthält iostat)
sudo apt install sysstat

# Debian/Ubuntu
sudo dnf install sysstat

# Fedora/RHEL
# Einfacher iostat-Aufruf
iostat
Linux 5.15.0 (hostname)    07/11/2025    _x86_64_    (4 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
□□   12.34    0.12    3.45    2.34    0.00   81.75

Device             tps    kB_read/s    kB_read    kB_wrtn/s    kB_wrtn
sda              15.67       123.45    1234567       456.78    4567890
```

Jede Spalte der `iostat`-Ausgabe erklärt

Für detaillierte Analysen verwendest du `iostat -x`:

```
iostat -x 1 5

# -x = erweiterte Statistiken
# 1  = jede Sekunde aktualisieren
# 5  = 5 Messungen, dann beenden
```

Ausgabe verstehen:

```
Device            r/s     w/s     rkB/s     wkB/s   rrqm/s   wrqm/s  %rrqm  %wrqm
avgrq-sz    avgqu-sz     await     r_await     w_await     svctm     %util
sda              12.34   45.67    123.45    456.78     0.12     2.34   0.97  4.87
25.6        0.85      14.5        8.2         16.8       2.1      12.2
```

Die wichtigsten Spalten erklärt:

Spalte	Bedeutung	Gute Werte	Schlechte Werte
r/s	Read-Operationen pro Sekunde	< 100 (HDD)	300 (HDD)
w/s	Write-Operationen pro Sekunde	< 50 (HDD)	200 (HDD)
rkB/s	Gelesene Kilobytes pro Sekunde	–	–
wkB/s	Geschriebene Kilobytes pro Sekunde	–	–
rrqm/s	Read-Requests merged pro Sekunde	5	< 1
wrqm/s	Write-Requests merged pro Sekunde	10	< 1
avgrq-sz	Durchschnittliche Request-Größe	32 KB	< 8 KB
avgqu-sz	Durchschnittliche Queue-Länge	< 2	10
await	Durchschnittliche Wartezeit (ms)	< 10 (SSD), < 20 (HDD)	50
%util	Auslastung der Festplatte	< 80%	95%

> □□ **Das brauchst du später, um:** zu verstehen, ob deine Festplatte der Flaschenhals ist und ob sich eine SSD lohnt.

Was bedeuten `%iowait` und `await` wirklich?

Diese beiden Werte verwirren viele Administratoren:

- **`%iowait` (aus `top` oder `iostat`):**
- Zeigt an, wie viel Zeit die CPU im Leerlauf verbringt, während mindestens ein Thread auf den Abschluss von Disk- oder Netzwerk-I/O wartet.
- *Wichtig:* `%iowait` ist kein direktes Maß für Festplattenauslastung. Ein hoher `%iowait` bei geringer sonstiger CPU-Last signalisiert einen klaren I/O-Bottleneck.
- **`await` (aus `iostat -x`):**
- Durchschnittliche Zeit in Millisekunden, die ein I/O-Request vom Eintreffen in der Block-Layer-Queue bis zum finalen Abschluss durch das Speichermedium benötigt.
- Beinhaltet die Wartezeit in der Warteschlange plus die eigentliche physische Servicezeit des Geräts (wichtigster Indikator für I/O-Latenz!).

□□ Praktisches Beispiel: Langsame Festplatte identifizieren:

Du startest eine Anwendung und sie lädt sehr langsam.

Schauen wir, ob die Festplatte schuld ist:

```
iostat -x 1
Device            r/s     w/s     rkB/s     wkB/s   await     %util
sda             234.5    12.3   2345.67    123.45    45.2      98.7
```

Diagnose:

- `r/s` = 234.5: Sehr viele Read-Operationen (> 100 ist viel für HDD)
- `await` = 45.2ms: Sehr hohe Wartezeit (> 20ms ist schlecht für HDD)
- `%util` = 98.7%: Festplatte ist praktisch zu 100% ausgelastet

Ursache finden:

```
# Welcher Prozess verursacht die I/O-Last?
sudo iotop -o
Total  DISK READ :     2.35 M/s | Total DISK WRITE :     123.45 K/s
Actual DISK READ:      2.35 M/s | Actual DISK WRITE:     123.45 K/s
TID    PRIO  USER      DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
1234   be/4  user      2.34 M/s      0.00 B/s  0.00 % 89.12 % find /home -name "*.jpg"
```

Lösung: Der `find`-Befehl durchsucht das gesamte Home-Verzeichnis und blockiert dabei die Festplatte.

> ⚠ **Häufiger Fehler:** Nur auf die Transferrate (rkB/s, wkB/s) schauen. Eine Festplatte kann auch durch viele kleine Zugriffe blockiert werden, auch wenn die Transferrate niedrig ist.

I/O-Request-Ablauf im Block-Layer:

1. **Anwendungs-Requests (I/O-Erzeugung)** Prozesse im Userspace stellen Lese- oder Schreibanfragen an das Dateisystem
2. **I/O-Warteschlange (Kernel Block Layer)** Der I/O-Scheduler reiht Operationen ein; avgqu-sz beziffert wartende Requests
3. **Speichermedium-Verarbeitung (Storage Controller)** NVMe/SSD bedient Anforderungen; %util zeigt die zeitliche Sättigung des Geräts
4. **I/O-Abschluss & Quittierung** await erfasst die Gesamtlatenz aus Wartezeit in der Queue und Zugriffszeit

> **Das brauchst du später, um:** I/O-Engpässe zu identifizieren, die richtige Festplatte für deine Anwendung zu wählen und Performance-Probleme systematisch zu diagnostizieren.

`vmstat` – Virtueller Speicher und System-Performance

`vmstat` (Virtual Memory Statistics) gibt dir einen umfassenden Überblick über die System-Performance. Es kombiniert CPU-, Memory-, I/O- und System-Informationen in einer kompakten Darstellung.

Alle Spalten der `vmstat`-Ausgabe im Detail

```
vmstat 1 5

# 1 = jede Sekunde aktualisieren
# 5 = 5 Messungen, dann beenden

procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 1234567  67890 4567890   0    0    12    45  123  456 12  3 84  1  0
 2  0      0 1230000  67890 4567890   0    0    15    48  134  467 15  4 80  1  0
```

Spalten-Erklärung:

`procs` (Prozesse):

Spalte	Bedeutung	Kritische Werte
r	Runnable processes (warten auf CPU)	Anzahl CPU-Kerne
b	Blocked processes (warten auf I/O)	2

`Memory` (Speicher in KB):

Spalte	Bedeutung	Kritische Werte
swpd	Verwendeter Swap-Speicher	0 = System swappt
free	Freier Speicher	< 10% vom Gesamt-RAM
buff	Buffer-Speicher	–
cache	Cache-Speicher	–

`swap` (Swap-Aktivität in KB/s):

Spalte	Bedeutung	Kritische Werte
si	Swap in (von Festplatte in RAM)	0 = Problem!
so	Swap out (von RAM auf Festplatte)	0 = Problem!

`io` (I/O-Aktivität in Blöcken/s):

Spalte	Bedeutung	Kritische Werte
bi	Blocks in (gelesen)	1000
bo	Blocks out (geschrieben)	1000

`system` (System-Aktivität):

Spalte	Bedeutung	Kritische Werte
in	Interrupts pro Sekunde	5000
cs	Context switches pro Sekunde	10000

`cpu` (CPU-Zeit in %):

Spalte	Bedeutung	Kritische Werte

us	User time	–
sy	System time	30%
id	Idle time	< 20%
wa	I/O wait	20%
st	Steal time (nur bei VMs)	5%

Wann ist Swapping problematisch?

Swapping ist einer der häufigsten Performance-Killer. Hier erkennst du es:

□□ Praktisches Beispiel:

System wird plötzlich sehr langsam:

- vmstat 1
- procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
- r b swpd free buff cache si so bi bo in cs us sy id wa st
- 3 2 567890 12345 23456 345678 234 456 123 234 567 890 45 25 15 15 0
- 4 3 678901 11234 23456 345678 345 567 234 345 678 901 50 30 10 10 0

Alarmsignale:

- swpd > 0: System verwendet Swap-Speicher
- si > 0: Daten werden von Swap zurück in RAM geladen
- so > 0: Daten werden aus RAM in Swap ausgelagert
- b > 0: Prozesse warten auf I/O (wegen Swap-Zugriff)
- wa = 15%: CPU wartet auf I/O (Swap ist langsam)

> **□□ Was passiert hier?** Das System hat zu wenig RAM und muss ständig Daten zwischen RAM und Festplatte hin- und herschieben. Das ist extrem langsam!

Sofort-Maßnahmen:

```
# Speicherfresser finden und beenden
top -o %MEM

# Swap-Cache leeren (Vorsicht!)
sudo swapoff -a && sudo swapon -a
```

> ⚠ **Warnung:** `swapoff -a` kann das System zum Absturz bringen, wenn nicht genug RAM frei ist!

Memory-Leak erkennen

Ein Memory-Leak zeigt sich in `vmstat` durch kontinuierlich sinkenden freien Speicher:

- vmstat 5
- procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
- r b swpd free buff cache si so bi bo in cs us sy id wa st
- 1 0 0 2000000 67890 1000000 0 0 5 10 100 200 10 2 87 1 0
- 1 0 0 1800000 67890 1000000 0 0 5 10 100 200 10 2 87 1 0
- 1 0 0 1600000 67890 1000000 0 0 5 10 100 200 10 2 87 1 0
- 1 0 0 1400000 67890 1000000 0 0 5 10 100 200 10 2 87 1 0

Diagnose: Der freie Speicher (`free`) sinkt kontinuierlich von 2 GB auf 1.4 GB, obwohl Cache und Buffer konstant bleiben. Ein Prozess „frisst“ Speicher!

Täter finden:

```
# Prozesse nach Speicherverbrauch sortieren
```

```
ps aux --sort=-%mem | head -10
```

> **Das brauchst du später, um:** Memory-Leaks frühzeitig zu erkennen, Swap-Probleme zu diagnostizieren und die Gesamt-Performance deines Systems zu bewerten.

Warum `vmstat 1` besser ist als `vmstat`

Typischer Fehler: Viele verwenden `vmstat` ohne Parameter und bekommen verfälschte Werte.

```
# FALSCH - zeigt Durchschnittswerte seit Systemstart
vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 1234567  67890 4567890   0    0     8    15   89  156  5  1 93  1  0

# RICHTIG - zeigt aktuelle Werte jede Sekunde
vmstat 1
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 3  2      0 1234567  67890 4567890   0    0   234   456  567  890 45 25 15 15  0
```

Der Unterschied: Ohne Parameter zeigt `vmstat` Durchschnittswerte seit dem letzten Systemstart. Bei einem Performance-Problem brauchst du aber die **aktuellen** Werte!

> **Das brauchst du später, um:** Performance-Probleme in Echtzeit zu diagnostizieren und nicht von historischen Durchschnittswerten verwirrt zu werden.

`netstat` und `ss` – Netzwerkverbindungen überwachen

Netzwerkverbindungen zu überwachen ist ein essentieller Teil der Linux-Administration. Du musst wissen, welche Dienste auf welchen Ports lauschen, welche Verbindungen aktiv sind und ob verdächtige Aktivitäten stattfinden.

Unterschied zwischen `netstat` und dem modernen `ss`

`netstat` ist das traditionelle Tool, das du wahrscheinlich zuerst lernst:

- Seit Jahrzehnten Standard auf Unix-Systemen
- Einfache Syntax und bekannte Parameter
- Wird langsam durch ss ersetzt

`ss` (Socket Statistics) ist der moderne Nachfolger:

- Deutlich schneller, besonders bei vielen Verbindungen
- Mehr Details und bessere Filteroptionen
- Standard in modernen Linux-Distributionen

> **Das brauchst du später, um:** beide Tools zu beherrschen, da du in der Praxis auf beide treffen wirst. Ältere Systeme haben oft nur netstat, neuere bevorzugen.

Alle Socket-Zustände (LISTEN, ESTABLISHED, etc.) erklärt

TCP-Verbindungen durchlaufen verschiedene Zustände. Diese zu verstehen hilft dir bei der Diagnose:

- netstat -tn
- Active Internet connections (w/o servers)
- Proto Recv-Q Send-Q Local Address Foreign Address State
- tcp 0 0 192.168.1.100:22 192.168.1.50:54321 ESTABLISHED
- tcp 0 0 192.168.1.100:80 203.0.113.10:45678 TIME_WAIT
- tcp 0 0 192.168.1.100:443 198.51.100.20:12345 CLOSE_WAIT

Die wichtigsten TCP-Zustände:

Zustand	Bedeutung	Normal?
LISTEN	Port wartet auf eingehende Verbindungen	✅ Ja
ESTABLISHED	Aktive, funktionierende Verbindung	✅ Ja
TIME_WAIT	Verbindung beendet, wartet auf Cleanup	✅ Ja, aber viele = Problem
CLOSE_WAIT	Verbindung wird beendet	⚠ Viele = Anwendungsproblem
FIN_WAIT1	Verbindung wird aktiv beendet	✅ Kurzzeitig normal
FIN_WAIT2	Warten auf Bestätigung der Beendigung	✅ Kurzzeitig normal
SYN_SENT	Verbindungsaufbau läuft	⚠ Viele = Netzwerkproblem
SYN_RECV	Verbindungsanfrage empfangen	⚠ Viele = möglicher Angriff

Häufige Probleme:

Zu viele TIME_WAIT-Verbindungen:

```
netstat -tn | grep TIME_WAIT | wc -l
15432
```

Problem: Anwendung öffnet zu viele Verbindungen zu schnell.

Viele CLOSE_WAIT-Verbindungen:

```
$ netstat -tn | grep CLOSE_WAIT | wc -l
```

```
234
```

Problem: Anwendung schließt Verbindungen nicht ordnungsgemäß.

□□ Praktisches Beispiel:

Offene Ports und verdächtige Verbindungen finden

Schritt 1: Alle lauschenden Ports auflisten

```
netstat -tlnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1234/sshd
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
5678/master
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
9012/apache2
tcp6       0      0 :::22                   :::*                    LISTEN      1234/sshd
tcp6       0      0 ::1:25                  :::*                    LISTEN
5678/master
```

Ausgabe verstehen:

- **Proto:** Protokoll (tcp, tcp6, udp, udp6)
- **Local Address:** IP:Port auf dem gelauscht wird
- **State:** Verbindungsstatus (LISTEN = wartet auf Verbindungen)
- **PID/Program name:** Welcher Prozess nutzt den Port

Wichtige Erkenntnisse:

- `Port 22` `SSH` ist auf allen Interfaces offen (0.0.0.0:22)
- `Port 25` `SMTP` nur lokal (127.0.0.1:25)

- `Port 80` `HTTP` ist öffentlich erreichbar (0.0.0.0:80)

Schritt 2: Unbekannte Ports identifizieren

```
# Port 1337 sollte nicht offen sein!
netstat -tlnp | grep :1337
tcp        0      0 0.0.0.0:1337            0.0.0.0:*               LISTEN
6666/suspicious
```

Schritt 3: Verdächtigen Prozess untersuchen

```
$ ps aux | grep 6666
user     6666  0.1  0.5  12345  6789 ?        S    14:30   0:01 /tmp/suspicious
```

Lösung:

```
# Verdächtig: Programm in /tmp!
ls -la /tmp/suspicious
-rwxr-xr-x 1 user user 123456 Jul 11 14:30 /tmp/suspicious
```

Schritt 4: Aktive Verbindungen zu diesem Port prüfen

```
netstat -tn | grep :1337
tcp        0      0 192.168.1.100:1337     203.0.113.50:54321     ESTABLISHED
```

> ⚠ **Warnung:** Ein unbekannter Prozess lauscht auf Port 1337 und hat eine aktive Verbindung zu einer externen IP. Das könnte Malware sein!

Moderne Alternative mit `ss`

```
# Entspricht netstat -tlnp
ss -tlnp

# Erweiterte Filter-Optionen
ss -t state established '( dport = :80 or dport = :443 )'

# Verbindungen zu einem bestimmten Port
ss -tn sport = :1337
```

Häufiger Fehler: `netstat` ohne Parameter verwenden

SCHLECHT – zeigt zu viel und ist langsam:

```
$ netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 hostname.local:ssh      client.local:54321     ESTABLISHED
```

```
tcp        0      0 hostname.local:http     google.com:80          TIME_WAIT
...
(hunderte von Zeilen)
```

Probleme:

- DNS-Auflösung macht die Ausgabe langsam
- Zu viele Informationen, unübersichtlich
- Keine Prozess-Informationen

BESSER – gezielt nach dem suchen, was du brauchst:

```
# Nur lauschende TCP-Ports mit Prozess-Info, numerisch
$ netstat -tlnp

# Oder mit ss (moderner)
$ ss -tlnp
```

Netzwerk-Monitoring-Checkliste:

Tägliche Checks:

```
# 1. Welche Dienste lauschen?
ss -tlnp | grep LISTEN

# 2. Ungewöhnliche Verbindungen?
ss -tn | grep -v "127.0.0.1\|::1"

# 3. Zu viele TIME_WAIT?
ss -tn state time-wait | wc -l
```

```
# 4. Verdächtige Ports?
ss -tlnp | grep -E ":(1337|4444|6666|31337)"
```

Bei Performance-Problemen:

```
# Verbindungen pro Zustand zählen
ss -tn | awk '{print $1}' | sort | uniq -c

# Top-Verbindungen nach Remote-IP
ss -tn | awk '{print $4}' | cut -d: -f1 | sort | uniq -c | sort -nr | head -10
```

> **Das brauchst du später, um:** Netzwerkprobleme zu diagnostizieren, Sicherheitsvorfälle zu erkennen und die Netzwerk-Performance zu überwachen. Und als Administrator verdächtige Netzwerkaktivitäten zu erkennen, Performance-Probleme durch zu viele Verbindungen zu identifizieren und Netzwerkdienste zu überwachen.

Log-Dateien systematisch analysieren

Das systemd-Journal verstehen

Das systemd-Journal ist das moderne Logging-System von Linux und hat die traditionellen Log-Dateien in `/var/log` weitgehend ersetzt. Als Linux-Administrator musst du `journalctl` beherrschen – es ist dein wichtigstes Werkzeug zur Analyse von Systemproblemen.

`journalctl` – Dein wichtigstes Log-Tool

`journalctl` ist der Befehl, mit dem du auf das systemd-Journal zugreifst. Im Gegensatz zu traditionellen Log-Dateien speichert das Journal strukturierte Daten, die du gezielt filtern und durchsuchen kannst.

Grundlegende Verwendung:

```
# Alle Journal-Einträge anzeigen (älteste zuerst)
journalctl

# Neueste Einträge zuerst (wie tail -f)
journalctl -r

# Nur die letzten 20 Zeilen
journalctl -n 20

# Live-Modus: Neue Einträge in Echtzeit verfolgen
journalctl -f
```

Warum ist `journalctl` so wichtig? Traditionelle Log-Dateien sind nur Text-Dateien. Das systemd-Journal speichert zusätzliche Metadaten wie Zeitstempel, Prioritäten, Prozess-IDs und Service-Namen. Das macht die Analyse viel präziser.

> □□ **Das brauchst du später, um:** Systemprobleme schnell zu diagnostizieren, ohne durch hunderte von Log-Zeilen scrollen zu müssen.

Wichtigste Parameter: `-f`, `-u`, `--since`, `--until`

Live-Monitoring mit `-f`:

```
# Neue Log-Einträge in Echtzeit verfolgen
journalctl -f

# Kombiniert mit anderen Filtern
journalctl -f -u ssh.service
```

Service-spezifische Logs mit `-u`:

```
# Nur Logs vom SSH-Service
journalctl -u ssh.service

# Mehrere Services gleichzeitig
journalctl -u ssh.service -u apache2.service

# Alle Services mit "network" im Namen
journalctl -u "*network*"
```

Zeitbasierte Filter mit `--since` und `--until`:

```
# Logs der letzten Stunde
journalctl --since "1 hour ago"

# Logs von heute
journalctl --since "today"

# Logs von gestern
journalctl --since "yesterday" --until "today"

# Spezifischer Zeitraum
journalctl --since "2025-07-12 10:00:00" --until "2025-07-12 12:00:00"

# Logs der letzten 10 Minuten
journalctl --since "10 minutes ago"
```

Kombinierte Filter:

```
# SSH-Logs der letzten 2 Stunden
journalctl -u ssh.service --since "2 hours ago"

# Fehler-Logs von heute
journalctl -p err --since "today"

# Live-Monitoring nur für kritische Fehler
journalctl -f -p crit
```

Prioritäts-Level verstehen:

```
# Nur Fehler und kritische Meldungen
journalctl -p err

# Alle Prioritäten von "warning" aufwärts
journalctl -p warning
```

Priorität	Nummer	Bedeutung	Wann verwenden?
emerg	0	System ist unbenutzbar	Systemausfall
alert	1	Sofortiges Handeln erforderlich	Kritische Fehler
crit	2	Kritische Bedingungen	Hardware-Probleme
err	3	Fehler-Bedingungen	Service-Fehler
warning	4	Warnungen	Potentielle Probleme
notice	5	Normale, aber wichtige Ereignisse	Service-Starts
info	6	Informative Nachrichten	Normale Aktivität

Priorität	Nummer	Bedeutung	Wann verwenden?
debug	7	Debug-Nachrichten	Entwicklung

□□ Praktisches Beispiel:

Systemstart-Probleme analysieren

Stell dir vor, dein System startet langsam oder ein Service funktioniert nicht.

Hier ist die systematische Vorgehensweise:

Schritt 1: Boot-Prozess analysieren

```
# Alle Logs vom letzten Boot
journalctl -b

# Logs vom vorletzten Boot (falls System neu gestartet wurde)
journalctl -b -1

# Nur Fehler beim letzten Boot
journalctl -b -p err
```

Schritt 2: Zeitaufwändige Services identifizieren

```
# Boot-Zeit-Analyse
systemd-analyze blame
Startup finished in 2.547s (kernel) + 8.234s (userspace) = 10.781s
graphical.target reached after 8.234s in userspace

□ 3.456s NetworkManager.service
□ 2.123s mysql.service
□ 1.789s apache2.service
```

```
 0.987s ssh.service
```

Schritt 3: Problematischen Service genauer untersuchen

```
# MySQL-Service beim letzten Boot
journalctl -b -u mysql.service

# Ausgabe könnte so aussehen:
Jul 12 14:30:15 hostname systemd[1]: Starting MySQL Community Server...
Jul 12 14:30:16 hostname mysqld[1234]: [Warning] World-writable config file
'/etc/mysql/my.cnf' is ignored
Jul 12 14:30:17 hostname mysqld[1234]: [ERROR] Can't start server: Bind on TCP/IP port:
Address already in use
Jul 12 14:30:17 hostname systemd[1]: mysql.service: Main process exited, code=exited,
status=1/FAILURE
Jul 12 14:30:17 hostname systemd[1]: mysql.service: Failed with result 'exit-code'.
```

Schritt 4: Ursache identifizieren

```
# Welcher Prozess blockiert Port 3306?
netstat -tlnp | grep :3306
tcp        0      0 0.0.0.0:3306            0.0.0.0:*               LISTEN
5678/old_mysql

# Alter MySQL-Prozess läuft noch!
ps aux | grep 5678
mysql    5678  0.1  2.3  123456  234567 ?      S    14:25   0:05 /usr/sbin/old_mysql
```

Lösung: Der alte MySQL-Prozess blockiert den Port. Nach dem Beenden startet der Service normal.

> ⚠ **Typische Stolperfalle:** Viele schauen nur auf die Fehlermeldung "Can't start server", aber die eigentliche Ursache steht in der Zeile davor: "Address already in use".

> □□ **Tipp:** Wie du das Journal dauerhaft speicherst: Standardmäßig speichert das systemd-Journal Logs nur temporär im RAM. Nach einem Neustart sind sie weg!

Problem prüfen:

```
# Ist das Journal persistent?
ls -la /var/log/journal/
ls: cannot access '/var/log/journal/': No such file or directory

# Oder existiert nur temporärer Speicher?
ls -la /run/log/journal/
drwxr-sr-x+ 3 root systemd-journal 4096 Jul 12 14:30 .
```

Persistente Speicherung aktivieren:

```
# Verzeichnis erstellen
sudo mkdir -p /var/log/journal

# Korrekte Berechtigungen setzen
sudo chown root:systemd-journal /var/log/journal
sudo chmod 2755 /var/log/journal

# systemd-journald neu starten
sudo systemctl restart systemd-journald
```

```
# Prüfen, ob es funktioniert
ls -la /var/log/journal/
drwxr-sr-x+ 3 root systemd-journal 4096 Jul 12 14:35 .
drwxr-sr-x+ 3 root systemd-journal 4096 Jul 12 14:35 a1b2c3d4e5f6...
```

Alternativ: Konfiguration über `/etc/systemd/journald.conf`:

```
sudo nano /etc/systemd/journald.conf

# Diese Zeile ändern:
Storage=persistent

# Speicher-Limits setzen (optional)
SystemMaxUse=500M
RuntimeMaxUse=100M

# Konfiguration aktivieren
sudo systemctl restart systemd-journald
```

Warum ist das wichtig? Ohne persistente Speicherung verlierst du alle Logs bei einem Systemausfall – genau dann, wenn du sie am meisten brauchst!

Nützliche Journal-Verwaltung:

```
# Aktueller Speicherverbrauch
journalctl --disk-usage
Archived and active journals take up 234.5M in the file system.

# Alte Logs bereinigen (älter als 2 Wochen)
sudo journalctl --vacuum-time=2weeks
```

```
# Speicher-Limit durchsetzen (max. 500MB)
sudo journalctl --vacuum-size=500M

# Anzahl der Journal-Dateien begrenzen
sudo journalctl --vacuum-files=10
```

Häufiger Fehler: Viele verwenden `journalctl` ohne Parameter und bekommen eine unübersichtliche Ausgabe mit tausenden Zeilen. Nutze immer Filter!

> **Das brauchst du später, um:** Systemprobleme systematisch zu diagnostizieren, Service-Fehler zu identifizieren und Boot-Probleme zu lösen. In der LPIC-1-Prüfung ist `journalctl` ein häufiges Thema.

Traditionelle Log-Dateien in /var/log

Obwohl das systemd-Journal moderne Logging-Funktionen bietet, verwenden viele Dienste und Anwendungen weiterhin traditionelle Log-Dateien in `/var/log`. Als Linux-Administrator musst du beide Systeme beherrschen, da sie sich ergänzen.

Welche Datei enthält was? (syslog, auth.log, kern.log, etc.)

Das Verzeichnis `/var/log` enthält verschiedene spezialisierte Log-Dateien. Jede hat einen bestimmten Zweck:

- ls -la /var/log/
- rw-r----- 1 syslog adm 123456 Jul 12 14:30 syslog
- rw-r----- 1 syslog adm 45678 Jul 12 14:30 auth.log
- rw-r----- 1 syslog adm 67890 Jul 12 14:30 kern.log
- rw-r----- 1 syslog adm 34567 Jul 12 14:30 daemon.log
- rw-r----- 1 syslog adm 78901 Jul 12 14:30 mail.log
- rw-r--r-- 1 root root 23456 Jul 12 14:30 dpkg.log
- rw-r--r-- 1 root root 12345 Jul 12 14:30 apt/history.log

Die wichtigsten Log-Dateien im Überblick:

Datei	Inhalt	Wann verwenden?
syslog	Allgemeine Systemnachrichten	Überblick über Systemaktivität
auth.log	Authentifizierung und Autorisierung	Login-Probleme, SSH-Angriffe
kern.log	Kernel-Nachrichten	Hardware-Probleme, Treiber-Fehler
daemon.log	Nachrichten von System-Diensten	Service-spezifische Probleme
mail.log	E-Mail-System (Postfix, Sendmail)	Mail-Server-Diagnose
apache2/error.log	Apache-Webserver-Fehler	Website-Probleme
mysql/error.log	MySQL-Datenbank-Fehler	Datenbank-Probleme
dpkg.log	Paket-Installation und -Updates	Software-Installation verfolgen
apt/history.log	Paket-Updates	Software-Installation verfolgen

□□ Praktisches Beispiel: Wichtige System-Logs analysieren

```
# Allgemeine Systemaktivität
tail -f /var/log/syslog
Jul 12 14:30:15 hostname systemd[1]: Started Daily apt download activities.
Jul 12 14:30:16 hostname cron[1234]: (root) CMD (test -x /usr/sbin/anacron || ( cd / &&
run-parts --report /etc/cron.daily ))

# Authentifizierung und Login-Versuche
tail -f /var/log/auth.log
Jul 12 14:30:17 hostname sshd[5678]: Accepted publickey for user from 192.168.1.50 port
54321 ssh2
Jul 12 14:30:18 hostname sudo[9012]: user : TTY=pts/0 ; PWD=/home/user ; USER=root ;
COMMAND=/bin/systemctl status ssh

# Kernel und Hardware
tail -f /var/log/kern.log
```

```
Jul 12 14:30:19 hostname kernel: [12345.678901] usb 2-1: new high-speed USB device number
3 using ehci-pci
Jul 12 14:30:20 hostname kernel: [12345.789012] usb-storage 2-1:1.0: USB Mass Storage
device detected
```

> **Das brauchst du später, um:** schnell die richtige Log-Datei für ein spezifisches Problem zu finden, ohne alle Logs durchsuchen zu müssen.

Log-Rotation verstehen und konfigurieren

Log-Dateien würden ohne Rotation unbegrenzt wachsen und irgendwann die Festplatte füllen. Das System `logrotate` verhindert das automatisch.

Wie Log-Rotation funktioniert:

```
# Aktuelle Log-Datei
/var/log/syslog

# Nach der ersten Rotation
/var/log/syslog      (neue, leere Datei)
/var/log/syslog.1    (alte Inhalte, komprimiert)

# Nach der zweiten Rotation
/var/log/syslog      (neue, leere Datei)
/var/log/syslog.1    (gestern)
/var/log/syslog.2.gz (vorgestern, komprimiert)

# Nach mehreren Rotationen
/var/log/syslog
/var/log/syslog.1
/var/log/syslog.2.gz
/var/log/syslog.3.gz
/var/log/syslog.4.gz
```

Logrotate-Konfiguration verstehen:

```
# Hauptkonfiguration
cat /etc/logrotate.conf

# Globale Einstellungen
weekly
rotate 4
create
compress
include /etc/logrotate.d

# Service-spezifische Konfiguration
$ cat /etc/logrotate.d/rsyslog
/var/log/syslog
/var/log/mail.info
/var/log/mail.warn
/var/log/mail.err
/var/log/daemon.log
/var/log/kern.log
/var/log/auth.log
/var/log/user.log
{
□weekly
□missingok
□rotate 52
□compress
□delaycompress
□notifempty
□create 640 syslog adm
□postrotate
□□/usr/lib/rsyslog/rsyslog-rotate
□endscript
}
```

Wichtige Logrotate-Optionen:

Option	Bedeutung	Beispiel

weekly	Rotation jede Woche	`daily`, `monthly`
rotate 52	52 alte Versionen behalten	1 Jahr bei wöchentlicher Rotation
compress	Alte Logs komprimieren	Spart Speicherplatz
delaycompress	Erst ab der 2. Rotation komprimieren	Für laufende Prozesse
notifempty	Leere Dateien nicht rotieren	Verhindert unnötige Rotationen
create 640 syslog adm	Neue Datei mit Berechtigungen erstellen	Sicherheit und Zugriff
postrotate	Befehle nach Rotation ausführen	Service neu laden

Manuelle Log-Rotation testen:

```
# Logrotate für alle Konfigurationen ausführen
sudo logrotate /etc/logrotate.conf

# Nur für eine spezifische Konfiguration
sudo logrotate /etc/logrotate.d/rsyslog

# Test-Modus (zeigt was passieren würde)
sudo logrotate -d /etc/logrotate.conf

# Erzwungene Rotation (auch wenn nicht nötig)
sudo logrotate -f /etc/logrotate.d/rsyslog
```

> ⚠ **Warnung:** Ohne Log-Rotation können Log-Dateien die gesamte Festplatte füllen und das System zum Absturz bringen!

□□ Praktisches Beispiel:

Failed-Login-Versuche finden

Ein häufiger Administrationsfall: Du vermutest Brute-Force-Angriffe auf dein System und willst fehlgeschlagene Login-Versuche analysieren.

Schritt 1: Fehlgeschlagene SSH-Logins finden

```
# Alle fehlgeschlagenen SSH-Logins
grep "Failed password" /var/log/auth.log
Jul 12 14:25:30 hostname sshd[1234]: Failed password for root from 203.0.113.10 port 45678
ssh2
Jul 12 14:25:32 hostname sshd[1235]: Failed password for admin from 203.0.113.10 port
45679 ssh2
Jul 12 14:25:34 hostname sshd[1236]: Failed password for user from 203.0.113.10 port 45680
ssh2
Jul 12 14:25:36 hostname sshd[1237]: Failed password for invalid user test from
203.0.113.10 port 45681 ssh2
```

Schritt 2: Angreifer-IPs identifizieren

```
# Top-10-Angreifer-IPs
grep "Failed password" /var/log/auth.log | awk '{print $11}' | sort | uniq -c | sort -nr |
head -10
 234 203.0.113.10
  45 198.51.100.20
  23 192.0.2.30
  12 203.0.113.40
   8 198.51.100.50
```

Schritt 3: Zeitliche Verteilung analysieren

```
# Angriffe pro Stunde
grep "Failed password" /var/log/auth.log | awk '{print $1" "$2" "$3}' | cut -d: -f1 | sort
| uniq -c
  45 Jul 12 14
  67 Jul 12 15
 123 Jul 12 16
  89 Jul 12 17
```

Schritt 4: Benutzerkonten-Targeting erkennen

```
# Welche Benutzernamen werden angegriffen?
grep "Failed password" /var/log/auth.log | awk '{print $9}' | sort | uniq -c | sort -nr
 156 root
  67 admin
  34 user
  23 test
  12 guest
```

Schritt 5: Erfolgreiche Logins nach Angriffen prüfen

```
# Erfolgreiche Logins von verdächtigen IPs
$ grep "Accepted" /var/log/auth.log | grep "203.0.113.10"

# (Hoffentlich leer!)
# Alle erfolgreichen Logins heute
grep "Accepted" /var/log/auth.log | grep "$(date '+%b %d')"
Jul 12 14:30:17 hostname sshd[5678]: Accepted publickey for user from 192.168.1.50 port
54321 ssh2
```

Automatisiertes Monitoring-Skript:

```
#!/bin/bash
# Einfaches Brute-Force-Detection-Skript

LOGFILE="/var/log/auth.log"
THRESHOLD=10

echo "=== Brute-Force-Analyse ==="
echo "Datum: $(date)"
echo

# IPs mit mehr als $THRESHOLD fehlgeschlagenen Logins
echo "Verdächtige IPs (> $THRESHOLD Fehlversuche):"
grep "Failed password" $LOGFILE | \
awk '{print $11}' | \
sort | uniq -c | \
awk -v threshold=$THRESHOLD '$1 > threshold {print $1 " Versuche von " $2}' | \
sort -nr

echo
echo "Letzte 5 fehlgeschlagene Logins:"
grep "Failed password" $LOGFILE | tail -5
```

> **Häufiger Fehler:** Nur in `/var/log/auth.log` schauen und dabei übersehen, dass bei Log-Rotation die Angriffe in `/var/log/auth.log.1` oder älteren Dateien stehen könnten.

Bessere Suche über alle rotierten Logs:

```
# Alle auth.log-Dateien durchsuchen (auch rotierte)
zgrep "Failed password" /var/log/auth.log*
```

Präventive Maßnahmen basierend auf Log-Analyse:

```
# Fail2ban installieren und konfigurieren
sudo apt install fail2ban

# Verdächtige IPs manuell blockieren
sudo iptables -A INPUT -s 203.0.113.10 -j DROP

# SSH-Konfiguration härten
sudo nano /etc/ssh/sshd_config

# PermitRootLogin no
# MaxAuthTries 3
# AllowUsers user1 user2
```

Warnung: Lösche niemals einfach Logs

Typischer Einsteiger-Fehler:

```
# NIEMALS SO MACHEN!
sudo rm /var/log/syslog
sudo rm /var/log/auth.log
```

Warum das problematisch ist:

- Laufende Prozesse: Services schreiben möglicherweise noch in die Dateien
- Berechtigungen: Neue Dateien haben falsche Berechtigungen
- Forensik: Bei Sicherheitsvorfällen sind die Logs weg
- Compliance: Viele Vorschriften verlangen Log-Aufbewahrung

Richtige Methoden zum Log-Cleanup:

```
# Log-Datei leeren (aber nicht löschen)
sudo truncate -s 0 /var/log/syslog

# Oder mit cat
sudo cat /dev/null > /var/log/syslog

# Logrotate für sofortige Rotation verwenden
sudo logrotate -f /etc/logrotate.d/rsyslog

# Alte rotierte Logs löschen (sicher)
sudo find /var/log -name "*.log.*.gz" -mtime +30 -delete
```

Log-Größe überwachen:

```
# Größte Log-Dateien finden
sudo du -sh /var/log/* | sort -hr | head -10
2.3G    /var/log/journal
456M    /var/log/apache2
123M    /var/log/syslog
 67M    /var/log/auth.log
 45M    /var/log/kern.log

# Automatische Warnung bei großen Logs
find /var/log -name "*.log" -size +100M -exec ls -lh {} \;
```

> **Das brauchst du später, um:** Sicherheitsvorfälle zu analysieren, Compliance-Anforderungen zu erfüllen und Systemprobleme auch rückwirkend zu diagnostizieren. Um auch traditionelle Log-Dateien systematisch zu analysieren, Brute-Force-Angriffe zu erkennen und Log-Rotation richtig zu konfigurieren. Viele ältere Systeme und Anwendungen nutzen noch diese traditionellen Logs.

Log-Analyse mit Shell-Tools

Während `journalctl` perfekt für systemd-Logs ist, brauchst du für traditionelle Log-Dateien und komplexe Analysen die klassischen Shell-Tools. `grep`, `awk` und `sed` sind deine wichtigsten Werkzeuge zur Log-Analyse – sie sind auf jedem Linux-System verfügbar und extrem mächtig.

`grep`, `awk`, `sed` für Log-Analyse nutzen

`grep` – Zeilen finden und filtern:

`grep` ist dein wichtigstes Tool zum Durchsuchen von Log-Dateien. Es findet Zeilen, die bestimmte Muster enthalten.

```
# Einfache Textsuche
grep "error" /var/log/syslog
Jul 12 14:30:15 hostname systemd[1]: Failed to start some.service.
Jul 12 14:30:16 hostname kernel: [12345.678] USB disconnect, address 1

# Groß-/Kleinschreibung ignorieren
grep -i "ERROR" /var/log/syslog

# Zeilen zählen statt anzeigen
grep -c "Failed password" /var/log/auth.log
234

# Zeilennummern anzeigen
grep -n "kernel" /var/log/syslog
1234:Jul 12 14:30:16 hostname kernel: [12345.678] USB disconnect

# Kontext anzeigen (2 Zeilen vor und nach dem Treffer)
grep -C 2 "error" /var/log/syslog
```

Wichtige `grep`-Optionen:

Option	Bedeutung	Wann verwenden?

-i	Groß-/Kleinschreibung ignorieren	Fehlersuche
-v	Invertiert (Zeilen OHNE Muster)	Unerwünschte Einträge ausblenden
-c	Nur Anzahl der Treffer	Statistiken erstellen
-n	Zeilennummern anzeigen	Kontext in großen Dateien
-A 5	5 Zeilen nach dem Treffer	Folge-Ereignisse sehen
-B 5	5 Zeilen vor dem Treffer	Ursache-Ereignisse sehen
-C 5	5 Zeilen vor und nach dem Treffer	Vollständiger Kontext

`awk` – Spalten extrahieren und verarbeiten:

`awk` ist perfekt für strukturierte Log-Dateien, wo du bestimmte Spalten brauchst.

```
# Nur die IP-Adressen aus auth.log extrahieren
grep "Failed password" /var/log/auth.log | awk '{print $11}'
203.0.113.10
198.51.100.20
192.0.2.30

# Zeitstempel und IP kombinieren
grep "Failed password" /var/log/auth.log | awk '{print $1" "$2" "$3" - "$11}'
Jul 12 14:25:30 - 203.0.113.10
Jul 12 14:25:32 - 203.0.113.10

# Spalten mit Bedingungen filtern
awk '$5 > 90 {print "⚠ WARNUNG: " $1 " ist zu " $5 " voll!"}' <(df -h | tail -n +2)
⚠ WARNUNG: /dev/sda1 ist zu 95% voll!
```

`sed` – Text ersetzen und bearbeiten:

`sed` ist ideal zum Bereinigen und Formatieren von Log-Ausgaben.

```
# IP-Adressen anonymisieren
grep "Failed password" /var/log/auth.log | sed 's/[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-
9]\{1,3\}\.[0-9]\{1,3\}/XXX.XXX.XXX.XXX/g'

# Nur bestimmte Zeilen anzeigen (Zeile 10-20)
sed -n '10,20p' /var/log/syslog

# Leere Zeilen entfernen
sed '/^$/d' /var/log/some.log

# Mehrere Leerzeichen durch eines ersetzen
sed 's/  */ /g' /var/log/some.log
```

> □□ **Das brauchst du später, um:** komplexe Log-Analysen durchzuführen, Reports zu erstellen und Log-Daten für weitere Verarbeitung aufzubereiten.

□□ Praktisches Beispiel:

Top-10-Fehlerquellen identifizieren

Stell dir vor, dein System hat Performance-Probleme und du willst die häufigsten Fehlerquellen identifizieren. Hier ist eine systematische Analyse:

Schritt 1: Alle Fehlermeldungen sammeln

```
# Alle Zeilen mit "error", "failed", "warning" aus syslog
grep -iE "(error|failed|warning)" /var/log/syslog > /tmp/errors.log

# Auch aus anderen wichtigen Logs
grep -iE "(error|failed|warning)" /var/log/kern.log >> /tmp/errors.log
grep -iE "(error|failed|warning)" /var/log/daemon.log >> /tmp/errors.log
```

Schritt 2: Fehlertypen kategorisieren

```
# Häufigste Fehlermeldungen (erste 5 Wörter)
awk '{print $5" "$6" "$7" "$8" "$9}' /tmp/errors.log | sort | uniq -c | sort -nr | head -
10
  45 Failed to start some.service
  23 kernel: USB disconnect, address
  18 systemd: Failed to reload
  12 NetworkManager: device eth0: link
   8 cron: (root) CMD failed
   6 apache2: [error] [client]
   4 mysql: [Warning] World-writable config
   3 postfix: warning: hostname verification
   2 sshd: error: Bind to
   1 cups: Job failed, will
```

Schritt 3: Zeitliche Verteilung analysieren\

```
# Fehler pro Stunde
awk '{print $3}' /tmp/errors.log | cut -d: -f1 | sort | uniq -c | sort -nr
  67 16
  45 15
  34 14
  23 17
  12 13
```

Schritt 4: Service-spezifische Analyse

```
# Welche Services verursachen die meisten Probleme?
awk '{print $5}' /tmp/errors.log | cut -d: -f1 | sort | uniq -c | sort -nr | head -5
  89 systemd
  45 kernel
  23 NetworkManager
  18 apache2
  12 cron
```


```
# Welche Services verursachen die meisten Probleme?
awk '{print $5}' /tmp/errors.log | cut -d: -f1 | sort | uniq -c | sort -nr | head -5
  89 systemd
  45 kernel
  23 NetworkManager
  18 apache2
  12 cron
```


Schritt 5: Detailanalyse der Top-Probleme

```
# Alle systemd-Fehler genauer analysieren
grep "systemd" /tmp/errors.log | awk '{for(i=6;i<=NF;i++) printf "%s ", $i; print ""}' |
sort | uniq -c | sort -nr | head -5
  34 Failed to start some.service: Unit some.service not found.
  23 Failed to reload apache2.service: Job for apache2.service failed.
  18 some.service: Main process exited, code=exited, status=1/FAILURE
  12 Timed out waiting for device dev-disk-by\x2duuid-12345.device.
   4 Failed to mount /mnt/backup: No such file or directory
```

One-Liner für häufige Log-Analysen

Hier sind bewährte One-Liner, die du als Administrator regelmäßig brauchst:

Sicherheits-Analysen:

```
# Top-10-Angreifer-IPs (SSH)
grep "Failed password" /var/log/auth.log | awk '{print $11}' | sort | uniq -c | sort -nr |
head -10

# Erfolgreiche Root-Logins
grep "Accepted.*root" /var/log/auth.log | awk '{print $1" "$2" "$3" von "$11}'

# Sudo-Befehle der letzten 24h
grep "$(date '+%b %d')" /var/log/auth.log | grep "sudo.*COMMAND" | awk
'{for(i=14;i<=NF;i++) printf "%s ", $i; print ""}'

# Ungewöhnliche Login-Zeiten (nachts)
grep "Accepted" /var/log/auth.log | awk '$3 ~ /^0[0-5]:/ {print $0}'
```

Performance-Analysen:

```
# Services, die am häufigsten neu gestartet werden
grep "Started|Stopped" /var/log/syslog | awk '{print $6}' | sort | uniq -c | sort -nr |
head -10

# Kernel-Fehler der letzten Woche
grep "$(date -d '7 days ago' '+%b %d')" /var/log/kern.log | grep -i error

# Speicher-Warnungen
grep -i "out of memory|oom|killed process" /var/log/kern.log

# Festplatten-Fehler
grep -i "i/o error|read error|write error" /var/log/kern.log
```

System-Monitoring:

```
# Häufigste Cron-Jobs
grep "CRON" /var/log/syslog | awk '{print $6}' | sort | uniq -c | sort -nr | head -10

# Netzwerk-Interface-Probleme
grep -i "link|carrier|duplex" /var/log/kern.log | tail -20

# USB-Geräte-Aktivität
grep "usb" /var/log/kern.log | grep "$(date '+%b %d')" | tail -10

# Temperatur-Warnungen
grep -i "temperature|thermal|overheat" /var/log/kern.log
```

> **Typischer Fehler:** Logs ohne Zeitstempel-Filter durchsuchen

Häufiger Einsteiger-Fehler:

```
# SCHLECHT - durchsucht alle Logs seit Jahren
grep "error" /var/log/syslog
(tausende von Zeilen aus Monaten/Jahren)
```

BESSER – mit Zeitfilter arbeiten:

```
# Nur Logs von heute
```

```
grep "$(date '+%b %d')" /var/log/syslog | grep -i error

# Letzte 100 Zeilen, dann filtern
tail -100 /var/log/syslog | grep -i error

# Bestimmten Zeitraum
sed -n '/Jul 12 14:00/,/Jul 12 16:00/p' /var/log/syslog | grep -i error

# Mit awk für präzise Zeitfilter
awk '/Jul 12 14:/ && /error/ {print}' /var/log/syslog
```

Warum ist das wichtig?

Ohne Zeitfilter bekommst du:

- Irrelevante alte Einträge
- Überwältigende Datenmengen
- Langsame Ausführung bei großen Log-Dateien
- Verwirrung zwischen aktuellen und historischen Problemen

Erweiterte Zeitfilter-Techniken:

```
# Logs der letzten 2 Stunden (mit GNU date)
SINCE=$(date -d '2 hours ago' '+%b %d %H:')
grep "^$SINCE" /var/log/syslog

# Mehrere Tage kombinieren
grep -E "$(date '+%b %d')|$(date -d 'yesterday' '+%b %d')" /var/log/syslog

# Zeitraum zwischen zwei Zeitpunkten
awk '/Jul 12 09:00/,/Jul 12 17:00/' /var/log/syslog | grep -i error
```

□□ Praktisches Beispiel: Monitoring-Dashboard mit Shell-Tools

```
#!/bin/bash
# Einfaches Log-Analyse-Dashboard

echo "=== System-Log-Analyse $(date) ==="
echo

echo "□□Fehler der letzten Stunde:"
LAST_HOUR=$(date -d '1 hour ago' '+%b %d %H:')
grep "^$LAST_HOUR" /var/log/syslog | grep -iE "(error|failed|warning)" | wc -l
echo

echo "□□Top-5-Fehlerquellen heute:"
grep "$(date '+%b %d')" /var/log/syslog | grep -iE "(error|failed)" |
awk '{print $5}' | cut -d: -f1 | sort | uniq -c | sort -nr | head -5
echo

echo "□□SSH-Angriffe heute:"
grep "$(date '+%b %d')" /var/log/auth.log | grep "Failed password" |
awk '{print $11}' | sort | uniq -c | sort -nr | head -3
echo

echo "□□Kritische Kernel-Meldungen:"
grep "$(date '+%b %d')" /var/log/kern.log | grep -iE "(error|critical|fatal)" | tail -3
```

> ⚠ **Performance-Tipp:** Bei sehr großen Log-Dateien verwende zgrep für komprimierte Logs und tail -f für Live-Monitoring statt grep über die gesamte Datei.

Kombinierte Analyse-Pipeline:

```
# Komplexe Analyse in einer Pipeline
grep "$(date '+%b %d')" /var/log/syslog |
```

```
    grep -i error |
    awk '{print $5}' |
    cut -d: -f1 |
    sort | uniq -c |
    sort -nr |
    head -10 |
    awk '{printf "%-20s %s Fehlern", $2, $1}'
```

> **Das brauchst du später, um:** effizient durch große Log-Mengen zu navigieren, relevante Informationen schnell zu finden und automatisierte Log-Analysen zu erstellen. Wie auch, um komplexe Log-Analysen durchzuführen, Sicherheitsvorfälle zu untersuchen und Performance-Probleme systematisch zu diagnostizieren. Diese Shell-Tools sind auf jedem Linux-System verfügbar und bilden die Grundlage für professionelle Log-Analyse.

Automatisierte Überwachung einrichten

Einfache Monitoring-Skripte erstellen

Als Linux-Administrator willst du nicht ständig manuell `top`, `iostat` oder `journalctl` ausführen müssen. Automatisierte Monitoring-Skripte überwachen dein System kontinuierlich und warnen dich, bevor Probleme kritisch werden.

Bash-Skript: CPU-Auslastung überwachen

Beginnen wir mit einem einfachen Skript, das die CPU-Auslastung überwacht:

```
#!/bin/bash
# cpu_monitor.sh - Einfaches CPU-Monitoring
# Konfiguration
CPU_THRESHOLD=80
```

```
LOGFILE="/var/log/cpu_monitor.log"
HOSTNAME=$(hostname)

# Aktuelle Load auslesen
LOAD_1MIN=$(cat /proc/loadavg | awk '{print $1}')
CPU_CORES=$(nproc)

# Load in Prozent umrechnen
LOAD_PERCENT=$(echo "scale=2; $LOAD_1MIN / $CPU_CORES * 100" | bc)

# Zeitstempel für Log
TIMESTAMP=$(date '+%Y-%m-%d %H:%M:%S')

# Prüfung und Logging
if (( $(echo "$LOAD_PERCENT > $CPU_THRESHOLD" | bc -l) )); then
    echo "[$TIMESTAMP] ⚠ WARNUNG: CPU-Last bei ${LOAD_PERCENT}% (Schwellenwert:
${CPU_THRESHOLD}%)" | tee -a $LOGFILE

    # Top-5-Prozesse mit höchster CPU-Last
    echo "[$TIMESTAMP] Top-5-CPU-Verbraucher:" >> $LOGFILE
    ps aux --sort=-%cpu | head -6 | tail -5 >> $LOGFILE
    echo "---" >> $LOGFILE

    # Rückgabewert für weitere Aktionen
    exit 1
else
    echo "[$TIMESTAMP] CPU-Last normal: ${LOAD_PERCENT}%" >> $LOGFILE
    exit 0
fi
```

Skript ausführbar machen und testen:

```
# Skript erstellen und Berechtigungen setzen
sudo nano /usr/local/bin/cpu_monitor.sh
sudo chmod +x /usr/local/bin/cpu_monitor.sh

# Ersten Test durchführen
sudo /usr/local/bin/cpu_monitor.sh
```

```
[2025-07-12 16:30:15] CPU-Last normal: 23.45%

# Log-Datei prüfen
sudo tail -5 /var/log/cpu_monitor.log
[2025-07-12 16:30:15] CPU-Last normal: 23.45%
```

> **Das brauchst du später, um:** CPU-Probleme zu erkennen, bevor das System überlastet wird und Benutzer sich beschweren.

Schwellenwerte definieren und Aktionen auslösen

Monitoring ohne Aktionen ist nutzlos. Hier erweitern wir das Skript um verschiedene Schwellenwerte und Aktionen:

```
#!/bin/bash
# advanced_cpu_monitor.sh - Erweiterte CPU-Überwachung
# Konfiguration
WARNING_THRESHOLD=70
CRITICAL_THRESHOLD=90
LOGFILE="/var/log/cpu_monitor.log"
EMAIL="admin@example.com"

# Funktionen definieren
send_alert() {
    local level=$1
    local message=$2
    local timestamp=$(date '+%Y-%m-%d %H:%M:%S')

    # Log schreiben
    echo "[$timestamp] $level: $message" | tee -a $LOGFILE

    # E-Mail senden (falls mailutils installiert)
    if command -v mail >/dev/null 2>&1; then
        echo "$message" | mail -s "[$level] CPU-Alarm auf $(hostname)" $EMAIL
```

```
    fi
    }

    get_top_processes() {
    echo "Top-5-CPU-Verbraucher:"
    ps aux --sort=-%cpu | head -6 | tail -5 | awk '{printf "%-10s %5s%% %sn", $1, $3, $11}'
    }

    # Hauptlogik
    LOAD_1MIN=$(cat /proc/loadavg | awk '{print $1}')
    CPU_CORES=$(nproc)
    LOAD_PERCENT=$(echo "scale=1; $LOAD_1MIN / $CPU_CORES * 100" | bc)

    # Schwellenwerte prüfen
    if (( $(echo "$LOAD_PERCENT > $CRITICAL_THRESHOLD" | bc -l) )); then
    MESSAGE="KRITISCH: CPU-Last bei ${LOAD_PERCENT}% (Limit: ${CRITICAL_THRESHOLD}%)
    $(get_top_processes)"
    send_alert "CRITICAL" "$MESSAGE"

    # Notfall-Aktion: Ressourcen-intensive Prozesse identifizieren
    echo "[$timestamp] Notfall-Analyse gestartet" >> $LOGFILE
    iostat -x 1 3 >> $LOGFILE 2>&1

    exit 2

    elif (( $(echo "$LOAD_PERCENT > $WARNING_THRESHOLD" | bc -l) )); then
    MESSAGE="WARNUNG: CPU-Last bei ${LOAD_PERCENT}% (Warnschwelle: ${WARNING_THRESHOLD}%)
    $(get_top_processes)"
    send_alert "WARNING" "$MESSAGE"
    exit 1

    else

    # Nur bei Bedarf loggen (alle 10 Minuten)
    if [ $(($(date +%M) % 10)) -eq 0 ]; then
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] CPU-Last normal: ${LOAD_PERCENT}%" >> $LOGFILE
    fi
    exit 0
    fi
```

Erweiterte Konfiguration mit verschiedenen Aktionen:

Schwellenwert	Aktion	Zweck
< 70%	Normales Logging (alle 10 Min)	Baseline-Dokumentation
70-89%	Warnung + E-Mail + Top-Prozesse	Frühe Warnung
≥ 90%	Kritischer Alarm + Detailanalyse	Sofortmaßnahmen

□□ Praktisches Beispiel:

E-Mail senden bei hoher Load

Für E-Mail-Benachrichtigungen brauchst du ein funktionierendes Mail-System:

Schritt 1: Einfaches Mail-System einrichten

```
# mailutils installieren
sudo apt install mailutils postfix

# Debian/Ubuntu
sudo dnf install mailx postfix

# Fedora/RHEL
# Postfix für lokale Mails konfigurieren
sudo dpkg-reconfigure postfix

# Wähle: "Local only"
# System mail name: dein-hostname.local
```

Schritt 2: Mail-Funktion testen

```
# Test-Mail senden
```

```
echo "Test-Nachricht" | mail -s "Test-Betreff" root

# Mail-Queue prüfen
mailq

# Lokale Mails lesen
mail
```

Schritt 3: Monitoring-Skript mit Mail-Integration

```
#!/bin/bash
# mail_cpu_monitor.sh - CPU-Monitoring mit E-Mail

send_mail_alert() {
    local subject=$1
    local body=$2
    local recipient="root"

    # Lokaler Admin
    # Detaillierte System-Info sammeln
    {
        echo "Hostname: $(hostname)"
        echo "Zeitpunkt: $(date)"
        echo "Uptime: $(uptime)"
        echo ""
        echo "$body"
        echo ""
        echo "=== Aktuelle System-Info ==="
        echo "Load Average: $(cat /proc/loadavg)"
        echo "Memory: $(free -h | grep '^Mem:')"
        echo "Disk: $(df -h / | tail -1)"
        echo ""
        echo "=== Top-Prozesse ==="
        ps aux --sort=-%cpu | head -10
    } | mail -s "$subject" $recipient
}

# Hauptskript
```

```
LOAD_1MIN=$(cat /proc/loadavg | awk '{print $1}')
CPU_CORES=$(nproc)
LOAD_PERCENT=$(echo "scale=1; $LOAD_1MIN / $CPU_CORES * 100" | bc)

if (( $(echo "$LOAD_PERCENT > 80" | bc -l) )); then
send_mail_alert "⚠ Hohe CPU-Last auf $(hostname)"
"CPU-Auslastung: ${LOAD_PERCENT}%

Das System zeigt eine ungewöhnlich hohe CPU-Last.
Bitte prüfe die laufenden Prozesse und ergreife gegebenenfalls Maßnahmen."
fi
```

Tipp: Warum du Hysterese brauchst (Ein- und Ausschaltschwellen)

Problem ohne Hysterese:

```
# SCHLECHT - Alarm-Spam möglich
if [ $CPU_LOAD -gt 80 ]; then
send_alert "CPU hoch!"
fi
```

Was passiert: CPU schwankt zwischen 79% und 81% → Hunderte E-Mails!

Lösung mit Hysterese:

```
#!/bin/bash
# hysteresis_monitor.sh - Monitoring mit Hysterese

ALERT_FILE="/tmp/cpu_alert_active"
WARNING_THRESHOLD=80
RECOVERY_THRESHOLD=70
```

```
CURRENT_LOAD=$(get_cpu_load)

# Deine Load-Funktion

if [ $CURRENT_LOAD -gt $WARNING_THRESHOLD ]; then

# Alarm auslösen (nur wenn noch nicht aktiv)
    if [ ! -f $ALERT_FILE ]; then
        send_alert "CPU-Last kritisch: ${CURRENT_LOAD}%"
        touch $ALERT_FILE
        echo "$(date): Alarm aktiviert bei ${CURRENT_LOAD}%" >> $ALERT_FILE
    fi

elif [ $CURRENT_LOAD -lt $RECOVERY_THRESHOLD ]; then

# Alarm zurücksetzen (nur wenn aktiv)
    if [ -f $ALERT_FILE ]; then
        send_alert "CPU-Last wieder normal: ${CURRENT_LOAD}%"
        rm -f $ALERT_FILE
    fi
fi
```

Das Hysterese-Prinzip im Alerting:

- **Alarm auslösen (Trigger):** CPU-Last übersteigt 80 % über mindestens 3 aufeinanderfolgende Messintervalle.
- **Entwarnung (Recovery):** Entwarnungs-Benachrichtigungen werden erst versendet, wenn der Wert signifikant unter den Schwellenwert fällt (z. B. unter 70 % für 3 Zyklen).
- **Toleranzabstand (Hysterese):** Der Abstand von 10 % verhindert kontinuierliches Alarm-Flackern (*Alert Flapping*) bei natürlichen Lastschwankungen rund um die 80-%-Marke.

Warum ist das wichtig?:

- Verhindert Alarm-Spam bei schwankenden Werten
- Reduziert False-Positives durch kurze Spitzen
- Professionelles Monitoring arbeitet immer mit Hysterese

- Bessere Benutzererfahrung für Administratoren

Erweiterte Monitoring-Skripte

Memory-Monitoring:

```
#!/bin/bash
# memory_monitor.sh - RAM-Überwachung

get_memory_usage() {

# Verfügbarer Speicher in Prozent
local total=$(grep MemTotal /proc/meminfo | awk '{print $2}')
local available=$(grep MemAvailable /proc/meminfo | awk '{print $2}')
local used_percent=$(echo "scale=1; (($total - $available) / $total) * 100" | bc)
echo $used_percent
}

MEMORY_USAGE=$(get_memory_usage)
THRESHOLD=85

if (( $(echo "$MEMORY_USAGE > $THRESHOLD" | bc -l) )); then
{
echo "Speicherverbrauch kritisch: ${MEMORY_USAGE}%"
echo ""
echo "=== Memory-Details ==="
free -h
echo ""
echo "=== Top-Memory-Verbraucher ==="
ps aux --sort=-%mem | head -10
echo ""
echo "=== Swap-Status ==="
swapon --show
} | mail -s "⚠ Hoher Speicherverbrauch auf $(hostname)" root
fi
```

Disk-Space-Monitoring:

```
#!/bin/bash
# disk_monitor.sh - Festplattenspeicher-Überwachung

check_disk_usage() {

# Alle gemounteten Dateisysteme prüfen
    df -h | awk 'NR>1 {
        gsub(/%/, "", $5)
        if ($5 > 90) {
            print "KRITISCH: " $1 " ist zu " $5 "% voll (" $4 " frei)"
        } else if ($5 > 80) {
            print "WARNUNG: " $1 " ist zu " $5 "% voll (" $4 " frei)"
        }
    }'
}

DISK_ALERTS=$(check_disk_usage)

if [ -n "$DISK_ALERTS" ]; then
    {
        echo "Festplattenspeicher-Warnung:"
        echo "$DISK_ALERTS"
        echo ""
        echo "=== Vollständige Disk-Übersicht ==="
        df -h
        echo ""
        echo "=== Größte Dateien in /var/log ==="
        du -sh /var/log/* 2>/dev/null | sort -hr | head -5
    } | mail -s "⚠ Festplattenspeicher-Warnung auf $(hostname)" root
fi
```

> ⚠ **Häufiger Fehler:** Monitoring-Skripte ohne Fehlerbehandlung schreiben. Wenn `bc` nicht installiert ist oder `/proc/loadavg` nicht lesbar ist, schlägt das Skript fehl.

Robuste Fehlerbehandlung:

```
#!/bin/bash
# robust_monitor.sh - Monitoring mit Fehlerbehandlung
# Abhängigkeiten prüfen
check_dependencies() {
    local missing=""

    for cmd in bc awk mail; do
        if ! command -v $cmd >/dev/null 2>&1; then
            missing="$missing $cmd"
        fi
    done

    if [ -n "$missing" ]; then
        echo "FEHLER: Fehlende Befehle:$missing" >&2
        exit 1
    fi
}

# Sichere Load-Auslese
get_load_safe() {
    if [ -r /proc/loadavg ]; then
        cat /proc/loadavg | awk '{print $1}' 2>/dev/null
    else
        echo "0.0"
    fi
}

# Hauptskript
check_dependencies

LOAD=$(get_load_safe)
if [ "$LOAD" = "0.0" ]; then
    echo "WARNUNG: Konnte Load nicht auslesen" >&2
    exit 1
fi
```

> **Das brauchst du später, um:** zuverlässige Monitoring-Systeme zu erstellen, die auch bei Problemen funktionieren und dich rechtzeitig vor kritischen Situationen warnen.

Cron-Jobs für regelmäßige Checks

Monitoring-Skripte sind nur dann nützlich, wenn sie regelmäßig ausgeführt werden. Cron ist der Standard-Scheduler in Linux und ermöglicht es dir, deine Monitoring-Skripte automatisch zu bestimmten Zeiten laufen zu lassen.

Monitoring-Skripte zeitgesteuert ausführen

Grundlagen der Cron-Syntax:

```
# Crontab-Format:
# Minute Stunde Tag Monat Wochentag Befehl
#   0-59  0-23  1-31  1-12   0-7
# Beispiele:
# */5 * * * *     - Alle 5 Minuten
# 0 */2 * * *     - Alle 2 Stunden
# 0 9 * * 1-5     - Täglich um 9:00, Montag bis Freitag
# 30 2 * * 0      - Sonntags um 2:30
# 0 0 1 * *       - Am 1. jeden Monats um Mitternacht
```

Crontab bearbeiten und verwalten:

```
# Eigene Crontab bearbeiten
$ crontab -e

# Aktuelle Crontab anzeigen
```

```
$ crontab -l

# Root-Crontab bearbeiten (für System-Monitoring)
$ sudo crontab -e

# Crontab eines anderen Benutzers bearbeiten
$ sudo crontab -e -u username
```

□□ Praktisches Beispiel: Monitoring-Crontab einrichten

```
# System-Crontab für Monitoring (sudo crontab -e)
# CPU-Check alle 5 Minuten
*/5 * * * * /usr/local/bin/cpu_monitor.sh >/dev/null 2>&1

# Memory-Check alle 10 Minuten
*/10 * * * * /usr/local/bin/memory_monitor.sh >/dev/null 2>&1

# Disk-Check alle 30 Minuten
*/30 * * * * /usr/local/bin/disk_monitor.sh >/dev/null 2>&1

# Vollständiger System-Check täglich um 6:00
0 6 * * * /usr/local/bin/daily_system_check.sh

# Log-Cleanup wöchentlich sonntags um 3:00
0 3 * * 0 /usr/local/bin/log_cleanup.sh
```

> □□ **Das brauchst du später, um:** dein System automatisch zu überwachen, ohne ständig manuell Befehle eingeben zu müssen.

Log-Rotation für Monitoring-Daten

Monitoring-Skripte erzeugen eigene Log-Dateien, die ohne Rotation die Festplatte füllen können. Hier richtest du Log-Rotation für deine Monitoring-Logs ein:

Logrotate-Konfiguration für Monitoring:

```
# Datei erstellen: /etc/logrotate.d/monitoring
$ sudo nano /etc/logrotate.d/monitoring

# Inhalt:
/var/log/cpu_monitor.log
/var/log/memory_monitor.log
/var/log/disk_monitor.log
/var/log/system_check.log
{
    daily
    missingok
    rotate 30
    compress
    delaycompress
    notifempty
    create 640 root adm
    postrotate

# Optional: Monitoring-Service benachrichtigen
        /bin/kill -HUP $(cat /var/run/rsyslogd.pid 2>/dev/null) 2>/dev/null || true
    endscript
}
```

Logrotate-Konfiguration verstehen:

Option	Bedeutung	Warum wichtig?
daily	Tägliche Rotation	Monitoring-Logs wachsen schnell
rotate 30	30 Tage aufbewahren	Ausreichend für Trend-Analyse
compress	Alte Logs komprimieren	Speicherplatz sparen
delaycompress	Erst ab 2. Rotation komprimieren	Für laufende Prozesse
notifempty	Leere Dateien nicht rotieren	Unnötige Rotationen vermeiden

Option	Bedeutung	Warum wichtig?
create 640 root adm	Neue Datei mit korrekten Rechten	Sicherheit gewährleisten

Logrotate manuell testen:

```
# Test-Modus (zeigt was passieren würde)
sudo logrotate -d /etc/logrotate.d/monitoring

# Erzwungene Rotation zum Testen
sudo logrotate -f /etc/logrotate.d/monitoring

# Prüfen ob es funktioniert hat
$ ls -la /var/log/cpu_monitor.log*
-rw-r----- 1 root adm    1234 Jul 12 16:30 cpu_monitor.log
-rw-r----- 1 root adm    5678 Jul 11 16:30 cpu_monitor.log.1.gz
```

□□ Praktisches Beispiel:

Täglicher System-Health-Report

Hier erstellen wir ein umfassendes Skript, das täglich einen System-Health-Report per E-Mail sendet:

```
#!/bin/bash
# daily_system_check.sh - Täglicher System-Health-Report
# Konfiguration
REPORT_FILE="/tmp/system_report_$(date +%Y%m%d).txt"
EMAIL="admin@localhost"
HOSTNAME=$(hostname)

# Report-Header erstellen
```

```
create_header() {
    cat << EOF > $REPORT_FILE
========================================
TÄGLICHER SYSTEM-HEALTH-REPORT
========================================
Hostname: $HOSTNAME
Datum: $(date '+%Y-%m-%d %H:%M:%S')
Uptime: $(uptime -p)

EOF
}

# CPU-Status prüfen
check_cpu() {
    echo "=== CPU-STATUS ===" >> $REPORT_FILE

    # Load Average
    LOAD_1MIN=$(cat /proc/loadavg | awk '{print $1}')
    CPU_CORES=$(nproc)
    LOAD_PERCENT=$(echo "scale=1; $LOAD_1MIN / $CPU_CORES * 100" | bc)

    echo "Load Average: $(cat /proc/loadavg)" >> $REPORT_FILE
    echo "CPU-Auslastung: ${LOAD_PERCENT}% (${CPU_CORES} Kerne)" >> $REPORT_FILE

    # Warnung bei hoher Last
    if (( $(echo "$LOAD_PERCENT > 80" | bc -l) )); then
        echo "⚠WARNUNG: Hohe CPU-Last!" >> $REPORT_FILE
    else
        echo "CPU-Last normal" >> $REPORT_FILE
    fi

    echo "" >> $REPORT_FILE
}

# Memory-Status prüfen
check_memory() {
    echo "=== SPEICHER-STATUS ===" >> $REPORT_FILE

    # Memory-Info auslesen
    local total=$(grep MemTotal /proc/meminfo | awk '{print $2}')
    local available=$(grep MemAvailable /proc/meminfo | awk '{print $2}')
    local used_percent=$(echo "scale=1; (($total - $available) / $total) * 100" | bc)

    echo "Speicher-Übersicht:" >> $REPORT_FILE
```

```
    free -h >> $REPORT_FILE
    echo "" >> $REPORT_FILE
    echo "Speicherverbrauch: ${used_percent}%" >> $REPORT_FILE

    # Warnung bei hohem Speicherverbrauch
    if (( $(echo "$used_percent > 85" | bc -l) )); then
        echo "⚠WARNUNG: Hoher Speicherverbrauch!" >> $REPORT_FILE
        echo "Top-5-Memory-Verbraucher:" >> $REPORT_FILE
        ps aux --sort=-%mem | head -6 | tail -5 >> $REPORT_FILE
    else
        echo "Speicherverbrauch normal" >> $REPORT_FILE
    fi

    echo "" >> $REPORT_FILE
}

# Disk-Status prüfen
check_disk() {
    echo "=== FESTPLATTEN-STATUS ===" >> $REPORT_FILE

    # Disk-Usage
    df -h >> $REPORT_FILE
    echo "" >> $REPORT_FILE

    # Warnungen für volle Festplatten
    local disk_warnings=$(df -h | awk 'NR>1 {gsub(/%/, "", $5); if ($5 > 90) print "
KRITISCH: " $1 " ist zu " $5 "% voll"; else if ($5 > 80) print "⚠ WARNUNG: " $1 " ist zu
" $5 "% voll"}')

    if [ -n "$disk_warnings" ]; then
        echo "$disk_warnings" >> $REPORT_FILE
    else
        echo "Alle Festplatten haben ausreichend Platz" >> $REPORT_FILE
    fi

    echo "" >> $REPORT_FILE
}

# Service-Status prüfen
check_services() {
    echo "=== WICHTIGE SERVICES ===" >> $REPORT_FILE

    # Liste wichtiger Services
    local services=("ssh" "cron" "rsyslog" "systemd-journald")
```

```
    for service in "${services[@]}"; do
        if systemctl is-active --quiet $service; then
            echo "✓ $service: Läuft" >> $REPORT_FILE
        else
            echo "✗ $service: Gestoppt oder fehlerhaft" >> $REPORT_FILE
        fi
    done

    echo "" >> $REPORT_FILE
}

# Letzte Fehler aus Logs
check_recent_errors() {
    echo "=== LETZTE SYSTEM-FEHLER ===" >> $REPORT_FILE

    # Fehler der letzten 24 Stunden
    local yesterday=$(date -d 'yesterday' '+%b %d')
    local today=$(date '+%b %d')

    echo "Kritische Fehler der letzten 24h:" >> $REPORT_FILE
    grep -E "($yesterday|$today)" /var/log/syslog | grep -iE "(error|critical|failed)" | tail
-10 >> $REPORT_FILE

    if [ $? -ne 0 ] || [ ! -s /tmp/recent_errors.tmp ]; then
        echo "Keine kritischen Fehler gefunden" >> $REPORT_FILE
    fi

    echo "" >> $REPORT_FILE
}

# Security-Check
check_security() {
    echo "=== SICHERHEITS-CHECK ===" >> $REPORT_FILE

    # Failed SSH-Logins der letzten 24h
    local failed_logins=$(grep "$(date '+%b %d')" /var/log/auth.log 2>/dev/null | grep "Failed
password" | wc -l)
    echo "Fehlgeschlagene SSH-Logins heute: $failed_logins" >> $REPORT_FILE

    if [ $failed_logins -gt 10 ]; then
        echo "⚠️ WARNUNG: Viele fehlgeschlagene Login-Versuche!" >> $REPORT_FILE
        echo "Top-Angreifer-IPs:" >> $REPORT_FILE
        grep "$(date '+%b %d')" /var/log/auth.log 2>/dev/null | grep "Failed password" | awk
```

```
    '{print $11}' | sort | uniq -c | sort -nr | head -5 >> $REPORT_FILE
    else
        echo "Normale Anzahl fehlgeschlagener Logins" >> $REPORT_FILE
    fi

    echo "" >> $REPORT_FILE
}

# Report-Footer
create_footer() {
    cat << EOF >> $REPORT_FILE
========================================
Report erstellt: $(date '+%Y-%m-%d %H:%M:%S')
Nächster Report: $(date -d 'tomorrow' '+%Y-%m-%d 06:00')
========================================
EOF
}

# Hauptfunktion
main() {

    # Report erstellen
    create_header
    check_cpu
    check_memory
    check_disk
    check_services
    check_recent_errors
    check_security
    create_footer

    # Report per E-Mail senden
    if command -v mail >/dev/null 2>&1; then
        mail -s "Täglicher System-Report: $HOSTNAME" $EMAIL < $REPORT_FILE
        echo "Report per E-Mail gesendet an: $EMAIL"
    else
        echo "WARNUNG: mail-Befehl nicht verfügbar. Report gespeichert in: $REPORT_FILE"
    fi

    # Report-Datei nach 7 Tagen löschen
    find /tmp -name "system_report_*.txt" -mtime +7 -delete 2>/dev/null
}

# Skript ausführen
```

```
main
```

Cron-Job für täglichen Report einrichten:

```
# In Root-Crontab eintragen (sudo crontab -e)
0 6 * * * /usr/local/bin/daily_system_check.sh >/dev/null 2>&1

# Oder mit Logging der Skript-Ausführung
0 6 * * * /usr/local/bin/daily_system_check.sh >> /var/log/system_check.log 2>&1
```

Warnung: Warum Monitoring-Skripte selbst überwacht werden müssen

Das Problem: Was passiert, wenn deine Monitoring-Skripte selbst fehlschlagen?

```
#!/bin/bash
# monitor_the_monitor.sh - Meta-Monitoring

MONITOR_LOG="/var/log/monitor_status.log"
LAST_RUN_FILE="/tmp/last_monitor_run"

check_monitor_health() {
    local script_name=$1
    local max_age_minutes=$2
    local script_log=$3

    # Prüfen ob das Skript kürzlich gelaufen ist
    if [ -f "$LAST_RUN_FILE" ]; then
        local last_run=$(cat $LAST_RUN_FILE)
        local current_time=$(date +%s)
        local age_minutes=$(( (current_time - last_run) / 60 ))
```

```
    if [ $age_minutes -gt $max_age_minutes ]; then
        echo "$(date): WARNUNG: $script_name läuft seit $age_minutes Minuten nicht!" >>
    $MONITOR_LOG
        return 1
    fi
else
    echo "$(date): WARNUNG: $script_name wurde noch nie ausgeführt!" >> $MONITOR_LOG
    return 1
fi

# Prüfen ob das Skript Fehler produziert
if [ -f "$script_log" ]; then
    local recent_errors=$(tail -100 $script_log | grep -i error | wc -l)
    if [ $recent_errors -gt 5 ]; then
        echo "$(date): WARNUNG: $script_name produziert viele Fehler ($recent_errors)!" >>
    $MONITOR_LOG
        return 1
    fi
fi

return 0
}

# Monitoring-Skripte prüfen
check_monitor_health "cpu_monitor.sh" 10 "/var/log/cpu_monitor.log"
check_monitor_health "memory_monitor.sh" 15 "/var/log/memory_monitor.log"
check_monitor_health "disk_monitor.sh" 35 "/var/log/disk_monitor.log"
```

Monitoring-Skripte mit Heartbeat:

```
#!/bin/bash
# cpu_monitor_with_heartbeat.sh - CPU-Monitor mit Heartbeat

HEARTBEAT_FILE="/tmp/cpu_monitor_heartbeat"

# Heartbeat aktualisieren
echo $(date +%s) > $HEARTBEAT_FILE
```

```
    # Normale Monitoring-Logik
    LOAD_1MIN=$(cat /proc/loadavg | awk '{print $1}')

    # ... Rest des Skripts
```

Cron-Jobs überwachen:

```
# Cron-Job-Status prüfen
sudo systemctl status cron

# Cron-Logs anzeigen
grep CRON /var/log/syslog | tail -10

# Letzte Cron-Ausführungen prüfen
sudo journalctl -u cron --since "1 hour ago"
```

> **Häufiger Fehler:** Monitoring-Skripte in Cron ohne absolute Pfade verwenden:

```
# FALSCH - PATH ist in Cron eingeschränkt
*/5 * * * * cpu_monitor.sh

# RICHTIG - Absolute Pfade verwenden
*/5 * * * * /usr/local/bin/cpu_monitor.sh

# Oder PATH in Crontab definieren
PATH=/usr/local/bin:/usr/bin:/bin
*/5 * * * * cpu_monitor.sh
```

Debugging von Cron-Jobs:

```
# Cron-Job mit Logging
*/5 * * * * /usr/local/bin/cpu_monitor.sh >> /var/log/cron_debug.log 2>&1

# Umgebung in Cron testen
* * * * * env > /tmp/cron_env.txt

# Manual testen mit Cron-Umgebung
sudo -u root env -i /bin/bash -c '/usr/local/bin/cpu_monitor.sh'
```

Robuste Cron-Job-Konfiguration:

```
# Vollständige Crontab mit Fehlerbehandlung
SHELL=/bin/bash
PATH=/usr/local/bin:/usr/bin:/bin
MAILTO=admin@localhost

# CPU-Monitor alle 5 Minuten
*/5 * * * * /usr/local/bin/cpu_monitor.sh || echo "CPU-Monitor fehlgeschlagen: $(date)" >>
/var/log/cron_errors.log

# Memory-Monitor alle 10 Minuten
*/10 * * * * /usr/local/bin/memory_monitor.sh || echo "Memory-Monitor fehlgeschlagen:
$(date)" >> /var/log/cron_errors.log

# Täglicher System-Check um 6:00
0 6 * * * /usr/local/bin/daily_system_check.sh || echo "System-Check fehlgeschlagen:
$(date)" >> /var/log/cron_errors.log

# Meta-Monitoring alle 30 Minuten
*/30 * * * * /usr/local/bin/monitor_the_monitor.sh
```

□□ **Das brauchst du später, um:** zuverlässige Monitoring-Systeme aufzubauen, die auch bei Problemen funktionieren und sich selbst überwachen. Wie auch um eine automatisierte Überwachung einzurichten, die zuverlässig läuft und dich rechtzeitig vor Problemen warnt. Cron-Jobs sind die Grundlage für professionelles System-Monitoring und ermöglichen es dir, proaktiv statt reaktiv zu administrieren.

Performance-Probleme systematisch lösen

Systematische Diagnose-Methode

Wenn dein Linux-System langsam läuft, ist der häufigste Fehler von Einsteigern: **Raten statt systematisch diagnostizieren**. Du öffnest wahllos verschiedene Tools, schaust hier und da, aber ohne klaren Plan. Das führt zu Verwirrung und verschwendeter Zeit.

Als professioneller Linux-Administrator brauchst du eine **strukturierte Methode**, um Performance-Probleme zu identifizieren und zu lösen. Hier lernst du die bewährte **USE-Methode** und einen praktischen **Diagnose-Workflow**.

Die USE-Methode: Utilization, Saturation, Errors

Die USE-Methode ist ein systematischer Ansatz zur Performance-Analyse, entwickelt von Brendan Gregg. Sie unterteilt jede Ressource in drei Kategorien:

`U` - Utilization (Auslastung):

- Wie viel Prozent der Ressource wird verwendet?
- Beispiel: CPU zu 80% ausgelastet

`S` - Saturation (Sättigung):

- Wie viel Arbeit wartet auf die Ressource?
- Beispiel: 10 Prozesse warten auf CPU-Zeit

`E` - Errors (Fehler):

- Welche Fehler treten bei der Ressource auf?
- Beispiel: Festplatten-I/O-Fehler

Beispiel: Festplatten-I/O-Fehler

USE-Methode für die wichtigsten Linux-Ressourcen:

Ressource	Utilization	Saturation	Errors
CPU	`top` (%CPU), `vmstat` (us+sy)	`vmstat` (r), Load Average	`dmesg`, `/var/log/kern.log`
Memory	`free` (used%), `top` (%MEM)	`vmstat` (si+so), Swap-Usage	`dmesg` (OOM), `/var/log/kern.log`
Disk I/O	`iostat` (%util)	`iostat` (avgqu-sz)	`dmesg`, `iostat` (Fehler-Spalten)
Network	`iftop`, `netstat`	`netstat` (Recv-Q, Send-Q)	`dmesg`, `/var/log/kern.log`

Das brauchst du später, um: Performance-Probleme methodisch zu analysieren, statt ziellos verschiedene Tools auszuprobieren.

□□ Praktisches Beispiel:

Schritt-für-Schritt-Diagnose eines langsamen Systems

Stell dir vor, Benutzer beschweren sich: „Das System ist heute sehr langsam!“ Hier ist deine systematische Vorgehensweise:

Schritt 1: Schneller Überblick verschaffen

```
# Erste Orientierung - Was ist los?
$ uptime
 16:45:23 up 5 days,  2:15,  3 users,  load average: 4.23, 3.87, 2.45

# Load ist hoch! Bei 4 CPU-Kernen bedeutet 4.23 = 105% Auslastung
```

```
$ nproc
4
```

Erste Erkenntnis: System ist überlastet (Load > Anzahl CPU-Kerne)

Schritt 2: CPU-Analyse (USE-Methode)

```
# Utilization: Wie ist die CPU ausgelastet?
$ top -n 1 | head -3
top - 16:45:24 up 5 days,  2:15,  3 users,  load average: 4.23, 3.87, 2.45
Tasks: 234 total,   4 running, 230 sleeping,   0 stopped,   0 zombie
%Cpu(s): 85.2 us, 12.3 sy,  0.0 ni,  1.2 id,  1.3 wa,  0.0 hi,  0.0 si,  0.0 st
```

Analyse der CPU-Zeiten:

- `85.2% us` (user): Sehr hoch! Benutzer-Programme verbrauchen viel CPU
- `12.3% sy` (system): Normal
- `1.2% id` (idle): Sehr niedrig! CPU hat kaum Leerlauf
- `1.3% wa` (iowait): Niedrig, also kein I/O-Problem

Zwischenfazit: CPU-Problem, nicht I/O-Problem

Schritt 3: Saturation prüfen

```
# Saturation: Wie viele Prozesse warten auf CPU?
vmstat 1 3
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 6  0      0 1234567 123456 2345678   0    0     5    12  234  567 85 12  2  1  0
```

```
 7  0      0 1230000 123456 2345678   0    0     3    15  245  578 87 11  1  1  0
 5  0      0 1235000 123456 2345678   0    0     2    10  223  556 83 13  3  1  0
```

Analyse:

- `r` = 6-7: 6-7 Prozesse warten auf CPU (bei 4 Kernen = Überlastung!)
- `b` = 0: Keine Prozesse warten auf I/O
- `si/so` = 0: Kein Swapping

Zwischenfazit: CPU-Saturation bestätigt – zu viele Prozesse kämpfen um CPU-Zeit

Schritt 4: Verursacher identifizieren

```
# Welche Prozesse verbrauchen die meiste CPU?
top -o %CPU -n 1 | head -15
PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1234 user      20   0  567890 123456  45678 R  45.2   1.5   12:34.56 python3
   5678 user      20   0  234567  67890  23456 R  23.4   0.8    8:45.67 node
   9012 user      20   0  345678  89012  34567 R  18.7   1.1    6:23.45 java
   3456 user      20   0  123456  34567  12345 R  12.3   0.4    4:56.78 gcc
   7890 root      20   0   45678  12345   6789 S   3.2   0.2    1:23.45 systemd
```

Analyse:

- `python3`: 45.2% CPU - Hauptverursacher!
- `node`: 23.4% CPU - Zweitgrößter Verbraucher
- `java`: 18.7% CPU - Drittgrößter Verbraucher
- `gcc`: 12.3% CPU - Kompilierung läuft

Schritt 5: Detailanalyse der Verursacher

```
# Was macht der Python-Prozess?
ps aux | grep 1234
user     1234  45.2  1.5 567890 123456 ?     R    14:30  12:34 python3
/home/user/scripts/heavy_calculation.py

# Prozess-Details
cat /proc/1234/cmdline
python3/home/user/scripts/heavy_calculation.py

# Wie lange läuft der Prozess schon?
ps -o pid,etime,cmd -p 1234
  PID     ELAPSED CMD
 1234    02:15:23 python3 /home/user/scripts/heavy_calculation.py
```

Erkenntnisse:

- Python-Skript läuft seit über 2 Stunden
- Führt schwere Berechnungen durch
- Blockiert das System für andere Benutzer

Schritt 6: Errors prüfen

```
# Kernel-Fehler prüfen
$ dmesg | tail -20
[12345.678] No errors found

# System-Logs prüfen
$ journalctl --since "1 hour ago" -p err
-- No entries --
```

Ergebnis: Keine Hardware-Fehler, reines Performance-Problem

Diagnose-Workflow nach der USE-Methodik:

1. **Überblick & Basismetriken** uptime, Load Average und dmesg zur ersten schnellen Lagebeurteilung prüfen
2. **USE-Methodik anwenden** Utilization, Saturation und Errors für CPU, RAM, Storage I/O und Netzwerk analysieren
3. **Engpass isolieren** Flaschenhals lokalisieren: CPU-Sättigung, Speicherdruck (PSI/OOM) oder I/O-Wait
4. **Verursachenden Prozess identifizieren** PID, Thread-Gruppe, User und cgroup v2 Slice mit top, vmstat, iotop oder bpftrace ermitteln
5. **Gezielte Entstörung** Prozesspriorität anpassen (nice), Dienst drosseln oder Konfigurationsparameter nachjustieren

Praktische Diagnose-Checkliste

Phase 1: Schneller Überblick (30 Sekunden)

```
# System-Status
uptime

# Load Average prüfen
free -h

# Speicher-Übersicht
df -h

# Festplattenspeicher
```

Phase 2: Ressourcen-Analyse (2 Minuten)

```
# CPU-Analyse
top -n 1 | head -10

# Top-Prozesse
vmstat 1 3

# CPU-Saturation
# Memory-Analyse
free -h

# Speicher-Details
ps aux --sort=-%mem | head -10

# Memory-Fresser
# I/O-Analyse
iostat -x 1 3

# Disk-Performance
iotop -o -n 3

# I/O-intensive Prozesse
```

Phase 3: Detailanalyse (5 Minuten)

```
# Prozess-Details
ps aux | grep <PID>

# Prozess-Informationen
cat /proc/<PID>/cmdline

# Vollständige Kommandozeile
ls -la /proc/<PID>/fd/

# Offene Dateien
```

```
# System-Logs
journalctl --since "1 hour ago" -p warning
dmesg | tail -20
```

> **Typischer Fehler:** Symptome statt Ursachen behandeln

Häufiger Einsteiger-Fehler:

```
# FALSCH - Symptom behandeln
sudo killall python3

# Alle Python-Prozesse beenden
sudo reboot

# System neu starten
```

Warum das problematisch ist:

- Datenverlust: Laufende Arbeiten gehen verloren
- Keine Lösung: Problem tritt wieder auf
- Keine Lerneffekt: Ursache bleibt unbekannt

RICHTIG – Ursache analysieren:

```
# 1. Prozess analysieren
ps aux | grep 1234
cat /proc/1234/cmdline
```

```
ls -la /proc/1234/

# 2. Prozess-Verhalten verstehen
strace -p 1234 -c

# Systemcalls analysieren
lsof -p 1234

# Offene Dateien prüfen
# 3. Benutzer kontaktieren
who

# Wer ist angemeldet?
w

# Was machen die Benutzer?
# 4. Koordinierte Lösung
kill -TERM 1234

# Sanft beenden
kill -KILL 1234

# Nur wenn nötig
```

Erweiterte Diagnose-Techniken

CPU-Profiling für komplexe Probleme:

```
# CPU-Verbrauch über Zeit verfolgen
pidstat -p 1234 1 10

# Prozess 1234, 10 Messungen
# Alle Threads eines Prozesses
top -H -p 1234

# Thread-Ansicht
# CPU-Affinität prüfen
```

```
taskset -p 1234

# Auf welchen CPU-Kernen läuft der Prozess?
```

Memory-Leak-Diagnose:

```
# Memory-Verbrauch über Zeit
while true; do
    ps -o pid,vsz,rss,comm -p 1234
    sleep 10
done

# Memory-Maps analysieren
cat /proc/1234/smaps | grep -E "(Size|Rss|Pss)"
```

I/O-Pattern-Analyse:

```
# Welche Dateien werden zugegriffen?
lsof -p 1234

# I/O-Pattern verstehen
strace -p 1234 -e read,write,open,close -f
```

Dokumentation der Diagnose

Warum Dokumentation wichtig ist:

- Wiederholbare Probleme: Lösung ist bekannt
- Trend-Analyse: Probleme häufen sich?

- Team-Kommunikation: Kollegen können helfen
- Compliance: Nachweis für Audits

Einfache Diagnose-Dokumentation:

```
#!/bin/bash
# diagnose_log.sh - Diagnose dokumentieren

LOGFILE="/var/log/performance_diagnosis.log"
TIMESTAMP=$(date '+%Y-%m-%d %H:%M:%S')

{
echo "=== PERFORMANCE-DIAGNOSE $TIMESTAMP ==="
echo "Problem: $1"
echo "Benutzer: $USER"
echo ""
echo "System-Status:"
uptime
free -h
echo ""
echo "Top-Prozesse:"
ps aux --sort=-%cpu | head -10
echo ""
echo "Lösung: $2"
echo "Status: $3"
echo "========================"
echo ""
} >> $LOGFILE

# Verwendung:
# ./diagnose_log.sh "System langsam" "Python-Skript beendet" "Gelöst"
```

> ⚠ **Wichtiger Hinweis:** Bevor du Prozesse beendest, versuche immer zuerst herauszufinden, was sie tun und ob sie wichtig sind. Ein kill -9 kann Daten zerstören oder wichtige Systemfunktionen unterbrechen.

Sichere Prozess-Beendigung:

```
# 1. Sanft beenden (SIGTERM)
kill -TERM 1234
sleep 10

# 2. Prüfen ob beendet
ps -p 1234 >/dev/null && echo "Läuft noch" || echo "Beendet"

# 3. Nur wenn nötig: Forciert beenden (SIGKILL)
kill -KILL 1234
```

> **Das brauchst du später, um:** Performance-Probleme systematisch und professionell zu diagnostizieren, ohne das System zu beschädigen oder wichtige Daten zu verlieren. Auch als Linux-Administrator methodisch und zielgerichtet Performance-Probleme zu lösen, statt planlos verschiedene Tools auszuprobieren. Diese systematische Herangehensweise unterscheidet professionelle Administratoren von Anfängern und führt zu schnelleren, besseren Lösungen.

Praktische Übungen und Troubleshooting

Kompletter System-Health-Check

Als Linux-Administrator brauchst du eine strukturierte Routine, um den Zustand deines Systems regelmäßig zu überprüfen. Ein systematischer System-Health-Check hilft dir dabei, Probleme zu erkennen, bevor sie kritisch werden, und gibt dir einen vollständigen Überblick über die Systemgesundheit.

5-Minuten-Routine-Check

Dieser schnelle Check solltest du täglich oder bei Verdacht auf Probleme durchführen. Er gibt dir in wenigen Minuten einen Überblick über die wichtigsten Systemmetriken:

```
#!/bin/bash
# quick_health_check.sh - 5-Minuten-System-Check

echo "========================================"
echo "SCHNELLER SYSTEM-HEALTH-CHECK"
echo "========================================"
echo "Hostname: $(hostname)"
echo "Datum: $(date '+%Y-%m-%d %H:%M:%S')"
echo "Uptime: $(uptime -p)"
echo ""

# 1. Load Average und CPU
echo "=== CPU-STATUS ==="
LOAD_1MIN=$(cat /proc/loadavg | awk '{print $1}')
CPU_CORES=$(nproc)
LOAD_PERCENT=$(echo "scale=1; $LOAD_1MIN / $CPU_CORES * 100" | bc)

echo "Load Average: $(cat /proc/loadavg)"
echo "CPU-Kerne: $CPU_CORES"
echo "CPU-Auslastung: ${LOAD_PERCENT}%"

if (( $(echo "$LOAD_PERCENT > 80" | bc -l) )); then
    echo "⚠ WARNUNG: Hohe CPU-Last!"
    echo "Top-3-CPU-Verbraucher:"
    ps aux --sort=-%cpu | head -4 | tail -3
else
    echo "✓ CPU-Last normal"
fi
echo ""

# 2. Speicher-Status
echo "=== SPEICHER-STATUS ==="
free -h
echo ""
```

```
# Speicherverbrauch berechnen
TOTAL_MEM=$(grep MemTotal /proc/meminfo | awk '{print $2}')
AVAIL_MEM=$(grep MemAvailable /proc/meminfo | awk '{print $2}')
USED_PERCENT=$(echo "scale=1; (($TOTAL_MEM - $AVAIL_MEM) / $TOTAL_MEM) * 100" | bc)

echo "Speicherverbrauch: ${USED_PERCENT}%"

if (( $(echo "$USED_PERCENT > 85" | bc -l) )); then
    echo "⚠ WARNUNG: Hoher Speicherverbrauch!"
    echo "Top-3-Memory-Verbraucher:"
    ps aux --sort=-%mem | head -4 | tail -3
else
    echo "✓ Speicherverbrauch normal"
fi

# Swap-Check
SWAP_USED=$(free | grep Swap | awk '{print $3}')
if [ "$SWAP_USED" -gt 0 ]; then
    echo "⚠ WARNUNG: System verwendet Swap ($SWAP_USED KB)"
else
    echo "✓ Kein Swap verwendet"
fi
echo ""

# 3. Festplatten-Status
echo "=== FESTPLATTEN-STATUS ==="
df -h
echo ""

# Festplatten-Warnungen
DISK_WARNINGS=$(df -h | awk 'NR>1 {
    gsub(/%/, "", $5)
    if ($5 > 95) {
        print "KRITISCH: " $1 " ist zu " $5 "% voll"
    } else if ($5 > 85) {
        print "⚠WARNUNG: " $1 " ist zu " $5 "% voll"
    }
}')

if [ -n "$DISK_WARNINGS" ]; then
    echo "$DISK_WARNINGS"
else
    echo "✓ Alle Festplatten haben ausreichend Platz"
fi
```

```
echo ""

# 4. Wichtige Services
echo "=== SERVICE-STATUS ==="
SERVICES=("ssh" "cron" "rsyslog" "systemd-journald")

for service in "${SERVICES[@]}"; do
    if systemctl is-active --quiet $service 2>/dev/null; then
        echo "$service: Aktiv"
    else
        echo "$service: Inaktiv oder nicht vorhanden"
    fi
done
echo ""

# 5. Netzwerk-Grundcheck
echo "=== NETZWERK-STATUS ==="

# Ping zu Google DNS (falls Internet verfügbar)
if ping -c 1 8.8.8.8 >/dev/null 2>&1; then
    echo "✓ Internet-Verbindung: OK"
else
    echo "⚠ Keine Internet-Verbindung oder DNS-Problem"
fi

# Lauschende Ports prüfen
LISTENING_PORTS=$(ss -tlnp | grep LISTEN | wc -l)
echo "Lauschende Ports: $LISTENING_PORTS"
echo ""

# 6. Letzte kritische Ereignisse
echo "=== LETZTE KRITISCHE EREIGNISSE ==="
RECENT_ERRORS=$(journalctl --since "1 hour ago" -p err --no-pager -q | wc -l)
if [ "$RECENT_ERRORS" -gt 0 ]; then
    echo "⚠ $RECENT_ERRORS Fehler in der letzten Stunde"
    echo "Letzte 3 Fehler:"
    journalctl --since "1 hour ago" -p err --no-pager -q | tail -3
else
    echo "✓ Keine kritischen Fehler in der letzten Stunde"
fi
echo ""

echo "========================================="
echo "SYSTEM-HEALTH-CHECK ABGESCHLOSSEN"
```

```
echo "========================================"
```

> **Das brauchst du später, um:** schnell einen Überblick über den Systemzustand zu bekommen, ohne verschiedene Tools einzeln aufrufen zu müssen.

Wöchentliche Tiefenanalyse

Einmal pro Woche solltest du eine ausführlichere Analyse durchführen, die auch Trends und weniger offensichtliche Probleme aufdeckt:

```
#!/bin/bash
# weekly_deep_check.sh - Wöchentliche Tiefenanalyse

REPORT_FILE="/tmp/weekly_system_report_$(date +%Y%m%d).txt"

{
echo "========================================"
echo "WÖCHENTLICHE SYSTEM-TIEFENANALYSE"
echo "========================================"
echo "Hostname: $(hostname)"
echo "Datum: $(date '+%Y-%m-%d %H:%M:%S')"
echo "Kernel: $(uname -r)"
echo "Distribution: $(lsb_release -d 2>/dev/null | cut -f2 || echo 'Unbekannt')"
echo ""

# 1. Erweiterte CPU-Analyse
echo "=== ERWEITERTE CPU-ANALYSE ==="
echo "CPU-Info:"
lscpu | grep -E "(Model name|CPU(s)|Thread|Core|Socket)"
echo ""

echo "Load-Trend der letzten 15 Minuten:"
uptime
echo ""
```

```
echo "CPU-Zeit-Verteilung:"
vmstat 1 3 | tail -1
echo ""

# 2. Memory-Tiefenanalyse
echo "=== MEMORY-TIEFENANALYSE ==="
echo "Detaillierte Speicher-Info:"
free -h
echo ""

echo "Memory-Fragmentierung:"
cat /proc/buddyinfo | head -3
echo ""

echo "Top-10-Memory-Verbraucher:"
ps aux --sort=-%mem | head -11
echo ""

# 3. I/O-Performance-Analyse
echo "=== I/O-PERFORMANCE-ANALYSE ==="
echo "Festplatten-Performance (3 Messungen):"
iostat -x 1 3 | tail -n +4
echo ""

echo "Inode-Verbrauch:"
df -i
echo ""

# 4. Netzwerk-Analyse
echo "=== NETZWERK-ANALYSE ==="
echo "Netzwerk-Interfaces:"
ip addr show | grep -E "(inet |UP|DOWN)"
echo ""

echo "Lauschende Services:"
ss -tlnp | grep LISTEN
echo ""

echo "Aktive Verbindungen (Top-10):"
ss -tn | awk 'NR>1 {print $4}' | cut -d: -f1 | sort | uniq -c | sort -nr | head -10
echo ""

# 5. Security-Analyse
```

```
echo "=== SECURITY-ANALYSE ==="
echo "Failed SSH-Logins (letzte 7 Tage):"
FAILED_LOGINS=$(grep "Failed password" /var/log/auth.log* 2>/dev/null | grep "$(date -d '7
days ago' '+%b')" | wc -l)
echo "Anzahl: $FAILED_LOGINS"

if [ "$FAILED_LOGINS" -gt 50 ]; then
    echo "⚠WARNUNG: Viele fehlgeschlagene Login-Versuche!"
    echo "Top-Angreifer-IPs:"
    grep "Failed password" /var/log/auth.log* 2>/dev/null | grep "$(date -d '7 days ago'
'+%b')" | awk '{print $11}' | sort | uniq -c | sort -nr | head -5
fi
echo ""

echo "Sudo-Aktivitäten (letzte 7 Tage):"
SUDO_COUNT=$(grep "sudo.*COMMAND" /var/log/auth.log* 2>/dev/null | grep "$(date -d '7 days
ago' '+%b')" | wc -l)
echo "Anzahl sudo-Befehle: $SUDO_COUNT"
echo ""

# 6. System-Updates
echo "=== SYSTEM-UPDATE-STATUS ==="
if command -v apt >/dev/null 2>&1; then
    echo "Verfügbare Updates (Debian/Ubuntu):"
    apt list --upgradable 2>/dev/null | wc -l
elif command -v dnf >/dev/null 2>&1; then
    echo "Verfügbare Updates (Fedora/RHEL):"
    dnf check-update -q | wc -l
fi
echo ""

# 7. Log-Analyse
echo "=== LOG-ANALYSE ==="
echo "Kernel-Warnungen (letzte 7 Tage):"
dmesg -T | grep -i "warning|error" | tail -10
echo ""

echo "Systemd-Service-Fehler (letzte 7 Tage):"
journalctl --since "7 days ago" -p err --no-pager -q | tail -10
echo ""

# 8. Ressourcen-Trends
echo "=== RESSOURCEN-TRENDS ==="
echo "Größte Verzeichnisse in /var:"
```

```
    du -sh /var/* 2>/dev/null | sort -hr | head -10
    echo ""

    echo "Älteste Prozesse:"
    ps -eo pid,etime,comm --sort=etime | tail -10
    echo ""

    echo "========================================"
    echo "TIEFENANALYSE ABGESCHLOSSEN"
    echo "Report gespeichert in: $REPORT_FILE"
    echo "========================================"

    } | tee $REPORT_FILE

    # Report per E-Mail senden (falls konfiguriert)
    if command -v mail >/dev/null 2>&1; then
    mail -s "Wöchentlicher System-Report: $(hostname)" root < $REPORT_FILE
    echo "Report per E-Mail gesendet."
    fi

    # Alte Reports löschen (älter als 30 Tage)
    find /tmp -name "weekly_system_report_*.txt" -mtime +30 -delete 2>/dev/null
```

Automatisierung der wöchentlichen Analyse:

```
    # In Root-Crontab eintragen (sudo crontab -e)
    0 7 * * 1 /usr/local/bin/weekly_deep_check.sh >/dev/null 2>&1

    # Jeden Montag um 7:00 Uhr
```

Dokumentation der Ergebnisse

Warum Dokumentation wichtig ist:

- Trend-Erkennung: Probleme entwickeln sich oft schleichend
- Baseline-Werte: Was ist "normal" für dein System?

- Compliance: Nachweis für Audits und Berichte
- Team-Kommunikation: Kollegen können Entwicklungen nachvollziehen

Einfaches Monitoring-Dashboard erstellen:

```
#!/bin/bash
# create_dashboard.sh - Einfaches Text-Dashboard

DASHBOARD_FILE="/var/www/html/system_dashboard.html"
TEMP_FILE="/tmp/dashboard_temp.html"

# HTML-Header
cat << 'EOF' > $TEMP_FILE
<!DOCTYPE html>
<html>
<head>
    <title>System Dashboard</title>
    <meta charset="UTF-8">
    <meta http-equiv="refresh" content="300">
    <style>
        body { font-family: monospace; margin: 20px; background: #f5f5f5; }
        .container { max-width: 1200px; margin: 0 auto; }
        .section { background: white; margin: 10px 0; padding: 15px; border-radius: 5px; }
        .ok { color: green; }
        .warning { color: orange; }
        .critical { color: red; }
        .header { background: #333; color: white; text-align: center; padding: 20px; }
        pre { background: #f8f8f8; padding: 10px; overflow-x: auto; }
    </style>
</head>
<body>
    <div class="container">
        <div class="header">
            <h1>System Dashboard</h1>
            <p>Hostname: $(hostname) | Letzte Aktualisierung: $(date)</p>
        </div>
EOF

# System-Status hinzufügen
```

```
    cat << EOF >> $TEMP_FILE
    <div class="section">
        <h2>System-Übersicht</h2>
        <pre>$(uptime)</pre>
        <pre>$(free -h)</pre>
    </div>

    <div class="section">
        <h2>CPU-Status</h2>
        <pre>$(top -bn1 | head -5)</pre>
    </div>

    <div class="section">
        <h2>Festplatten-Status</h2>
        <pre>$(df -h)</pre>
    </div>

    <div class="section">
        <h2>Top-Prozesse (CPU)</h2>
        <pre>$(ps aux --sort=-%cpu | head -10)</pre>
    </div>

    <div class="section">
        <h2>Top-Prozesse (Memory)</h2>
        <pre>$(ps aux --sort=-%mem | head -10)</pre>
    </div>

    <div class="section">
        <h2>Netzwerk-Status</h2>
        <pre>$(ss -tlnp | grep LISTEN | head -10)</pre>
    </div>

    <div class="section">
        <h2>Letzte System-Ereignisse</h2>
        <pre>$(journalctl --since "1 hour ago" -p warning --no-pager -q | tail -10)</pre>
    </div>
    EOF

    # HTML-Footer
    cat << 'EOF' >> $TEMP_FILE
    </div>
    </body>
    </html>
    EOF
```

```
# Dashboard aktivieren
sudo mv $TEMP_FILE $DASHBOARD_FILE
sudo chown www-data:www-data $DASHBOARD_FILE 2>/dev/null || true
echo "Dashboard erstellt: $DASHBOARD_FILE"
```

Dashboard automatisch aktualisieren:

```
# Cron-Job für Dashboard-Update (alle 5 Minuten)
*/5 * * * * /usr/local/bin/create_dashboard.sh >/dev/null 2>&1
```

Monitoring-Daten in CSV-Format sammeln:

```
#!/bin/bash
# collect_metrics.sh - Metriken für Trend-Analyse sammeln

CSV_FILE="/var/log/system_metrics.csv"
TIMESTAMP=$(date '+%Y-%m-%d %H:%M:%S')

# Header erstellen (nur beim ersten Mal)
if [ ! -f "$CSV_FILE" ]; then
echo
"Timestamp,Load_1min,CPU_Percent,Memory_Percent,Disk_Root_Percent,Processes_Total,Processe
s_Running" > $CSV_FILE
fi

# Metriken sammeln
LOAD_1MIN=$(cat /proc/loadavg | awk '{print $1}')
CPU_CORES=$(nproc)
CPU_PERCENT=$(echo "scale=1; $LOAD_1MIN / $CPU_CORES * 100" | bc)
```

```
TOTAL_MEM=$(grep MemTotal /proc/meminfo | awk '{print $2}')
AVAIL_MEM=$(grep MemAvailable /proc/meminfo | awk '{print $2}')
MEMORY_PERCENT=$(echo "scale=1; (($TOTAL_MEM - $AVAIL_MEM) / $TOTAL_MEM) * 100" | bc)

DISK_PERCENT=$(df / | awk 'NR==2 {gsub(/%/, "", $5); print $5}')

PROCESSES_TOTAL=$(ps aux | wc -l)
PROCESSES_RUNNING=$(ps aux | awk '$8 ~ /^R/ {count++} END {print count+0}')

# Daten in CSV schreiben
echo
"$TIMESTAMP,$LOAD_1MIN,$CPU_PERCENT,$MEMORY_PERCENT,$DISK_PERCENT,$PROCESSES_TOTAL,$PROCES
SES_RUNNING" >> $CSV_FILE

# Alte Daten löschen (älter als 90 Tage)
if [ $(wc -l < $CSV_FILE) -gt 12960 ]; then

# 90 Tage * 24h * 6 (alle 10 Min)
tail -12960 $CSV_FILE > ${CSV_FILE}.tmp && mv ${CSV_FILE}.tmp $CSV_FILE
fi
```

CSV-Daten analysieren:

```
# Durchschnittliche CPU-Last der letzten 24h
$ tail -144 /var/log/system_metrics.csv | awk -F, '{sum+=$3; count++} END {print
"Durchschnittliche CPU-Last: " sum/count "%"}'

# Höchste Memory-Nutzung der letzten Woche
$ tail -1008 /var/log/system_metrics.csv | awk -F, 'BEGIN{max=0} {if($4>max) max=$4} END
{print "Höchste Memory-Nutzung: " max "%"}'

# Trend-Analyse (steigend/fallend)
$ tail -20 /var/log/system_metrics.csv | awk -F, '{print $3}' | awk '{if(NR==1) prev=$1;
else {if($1>prev) up++; else down++; prev=$1}} END {print "CPU-Trend: "
(up>down?"steigend":"fallend")}'
```

> **Wichtiger Hinweis:** System-Health-Checks ohne Baseline-Werte durchführen. Du musst wissen, was für dein System "normal" ist, um Abweichungen zu erkennen.

Baseline-Werte ermitteln:

```
#!/bin/bash
# establish_baseline.sh - Baseline-Werte ermitteln

echo "Ermittle Baseline-Werte für $(hostname)..."
echo "Sammle Daten über 1 Stunde (alle 5 Minuten)..."

BASELINE_FILE="/var/log/system_baseline.txt"

{
echo "========================================"
echo "SYSTEM-BASELINE für $(hostname)"
echo "Zeitraum: $(date) bis $(date -d '+1 hour')"
echo "========================================"
echo ""
} > $BASELINE_FILE

# 12 Messungen über 1 Stunde
for i in {1..12}; do
  {
    echo "=== Messung $i/12 - $(date) ==="
    echo "Load: $(cat /proc/loadavg | awk '{print $1}')"
    echo "CPU: $(top -bn1 | grep "Cpu(s)" | awk '{print $2}' | cut -d% -f1)"
    echo "Memory: $(free | grep Mem | awk '{printf "%.1f", ($3/$2)*100}')"
    echo "Disk: $(df / | awk 'NR==2 {gsub(/%/, "", $5); print $5}')"
    echo ""
  } >> $BASELINE_FILE

  echo "Messung $i/12 abgeschlossen..."
  [ $i -lt 12 ] && sleep 300

# 5 Minuten warten
done
```

```
# Baseline-Zusammenfassung
{
echo "========================================"
echo "BASELINE-ZUSAMMENFASSUNG"
echo "========================================"

# Durchschnittswerte berechnen
AVG_LOAD=$(grep "Load:" $BASELINE_FILE | awk '{sum+=$2; count++} END {printf "%.2f",
sum/count}')
AVG_MEMORY=$(grep "Memory:" $BASELINE_FILE | awk '{sum+=$2; count++} END {printf "%.1f",
sum/count}')
AVG_DISK=$(grep "Disk:" $BASELINE_FILE | awk '{sum+=$2; count++} END {printf "%.1f",
sum/count}')

echo "Durchschnittliche Load: $AVG_LOAD"
echo "Durchschnittliche Memory-Nutzung: $AVG_MEMORY%"
echo "Durchschnittliche Disk-Nutzung: $AVG_DISK%"
echo ""
echo "Empfohlene Schwellenwerte:"
echo "Load-Warnung: $(echo "$AVG_LOAD * 1.5" | bc)"
echo "Memory-Warnung: $(echo "$AVG_MEMORY + 20" | bc)%"
echo "Disk-Warnung: $(echo "$AVG_DISK + 15" | bc)%"
echo ""
echo "Baseline erstellt: $(date)"
echo "========================================"
} >> $BASELINE_FILE

echo "Baseline-Ermittlung abgeschlossen!"
echo "Ergebnisse in: $BASELINE_FILE"
```

> **Das brauchst du später, um:** fundierte Entscheidungen über Schwellenwerte zu treffen und echte Probleme von normalen Schwankungen zu unterscheiden. Um auch ein professionelles System-Monitoring aufzubauen, das dir hilft, Probleme frühzeitig zu erkennen und die Gesundheit deiner Linux-Systeme kontinuierlich zu überwachen. Diese strukturierten Checks sind die Grundlage für zuverlässige IT-Infrastrukturen.

Wichtigste Monitoring-Befehle

Bereich	Befehl	Zweck	Wichtigste Parameter
CPU	`top`	Live-Prozessüberwachung	`M` (Memory), `P` (CPU), `k` (kill)
CPU	`htop`	Benutzerfreundliche Alternative	`F9` (kill), `F5` (Baum-Ansicht)
CPU	`uptime`	Load Average anzeigen	–
Memory	`free -h`	Speicher-Übersicht	`-h` (human readable)
Memory	`vmstat 1 3`	Virtueller Speicher	`1 3` (1 Sek, 3 Messungen)
Disk	`df -h`	Festplattenspeicher	`-h` (human readable), `-i` (Inodes)
Disk	`iostat -x 1 3`	I/O-Performance	`-x` (erweitert), `1 3` (Intervall)
Network	`ss -tlnp`	Netzwerkverbindungen	`-t` (TCP), `-l` (Listen), `-n` (numerisch)
Network	`netstat -tlnp`	Alternative zu ss	Gleiche Parameter
Logs	`journalctl -f`	Live-Log-Verfolgung	`-f` (follow), `-u` (Unit), `--since`
Logs	`grep -i error /var/log/syslog`	Log-Durchsuchung	`-i` (ignore case), `-C 3` (Kontext)

Troubleshooting-Checkliste für den Notfall

Bei Performance-Problemen – Schritt für Schritt

Schneller Überblick (30 Sekunden)

```
uptime

# Load Average prüfen
free -h

# Speicher-Status
df -h

# Festplattenspeicher
```

Hauptproblem identifizieren (2 Minuten)

```
top -n 1

# Top-Prozesse anzeigen
vmstat 1 3

# CPU/Memory-Saturation
iostat -x 1 3

# I/O-Performance
```

Verursacher finden (2 Minuten)

```
ps aux --sort=-%cpu | head -10

# CPU-Fresser
ps aux --sort=-%mem | head -10
```

```
# Memory-Fresser
iotop -o

# I/O-intensive Prozesse
```

Logs prüfen (1 Minute)

```
journalctl --since "1 hour ago" -p err
dmesg | tail -20
```

Befehlsreferenz (Cheatsheet)

top / htop / btop | Interaktive Echtzeit-Überwachung von vCPU-, Speicher- und Prozessstatusfree -h | Schnelle Übersicht zu physischem RAM, gecachtem Speicher und Swap

$ `vmstat 1 5 | Gibt Berichte über Run-Queue (r, b), Speicherdruck, Swap und CPU-Zustände aus iostat -xz 1 | Zeigt detaillierte I/O-Statistiken pro Blockgerät mit Wartezeiten (%util, await) iotop -o | Zeigt Prozesse mit tatsächlicher Festplattenaktivität in Echtzeit df -h && df -i | Prüft Speicherplatzbelegung und freie Dateisystem-Inodes journalctl -u <Unit> -f | Verfolgt System- und Dienstprotokolle live im Follow-Modus journalctl -p err..emerg -b | Filtert Journal-Logs nach Fehlern und Notfällen seit aktuellem Boot dmesg -T --level=err,warn | Zeigt Kernel-Fehlermeldungen mit menschenlesbaren Zeitstempeln ss -s | Aggregierte Socket-Statistiken für TCP, UDP und Raw-Sockets sar -u 1 5 | Langzeit-Aktivitätsreport für CPU-Auslastung aus dem sysstat-Paket bpftrace -e '...' | Dynamisches Kernel-Tracing für System-Calls, Latenzen und I/O-Events`

Weiterführende Ressourcen

Fazit

Systemüberwachung und Performance-Monitoring sind der Schlüssel zum proaktiven Betrieb stabiler Linux-Infrastrukturen. Indem du CPU-Engpässe, Speicherdruck (OOM), I/O-Bottlenecks und Netzwerk-Anomalien mit Standardwerkzeugen wie `vmstat`, `iostat`, `free`, `iotop` und `journalctl` präzise analysieren kannst, behebst du Störungen methodisch, bevor sie die Verfügbarkeit deiner Dienste beeinträchtigen.

> **Praxis-Tipp:** Vertraue bei Performance-Problemen nie auf ein einzelnes Werkzeug. Kombiniere immer `vmstat 1` für die Gesamtsicht (Run-Queue, Context Switches, Swap), `iostat -xz 1` für Disk-Wartezeiten (`%util`) und `journalctl -p err` für Fehlermeldungen, um die tatsächliche Ursache zweifelsfrei zu isolieren.

Im finalen zehnten Teil unserer Administrations-Serie widmen wir uns dem Lebenszyklus des Systems: **Weiter geht es mit:** Linux Administration #10: Boot-Management und Systemstart