Linux Performans İzleme Araçları - SRE/DevOps Profesyonel Rehber

{: .no_toc }

Bu rehber, Linux sistemlerde performans izleme ve sorun giderme için kullanılan temel araçların detaylı kullanım kılavuzudur. {: .fs-6 .fw-300 }

İçindekiler

{: .no_toc .text-delta }

1. TOC {:toc}

---

vmstat - Virtual Memory Statistics

- Genel Bakış
- Kurulum
- Kullanım Örnekleri
- Önemli Metrikler
- Alarm Kriterleri

3. htop - Interactive Process Viewer

- Genel Bakış
- Kurulum
- Kullanım Örnekleri
- Önemli Kısayollar

4. dstat - Versatile Resource Statistics

- Genel Bakış
- Kurulum
- Kullanım Örnekleri
- Plugin Listesi

5. sar - System Activity Reporter

- Genel Bakış
- Kurulum ve Aktivasyon
- Veri Toplama Ayarları
- Kullanım Örnekleri
- Tarihsel Veri Analizi
- Alarm Kriterleri

6. iotop - I/O Monitor

- Genel Bakış

---

vmstat - Virtual Memory Statistics

Genel Bakış

`vmstat` CPU, bellek, disk I/O ve sistem aktivitesini raporlayan hafif bir araçtır.

Kullanım Örnekleri

```
# Temel kullanım - 2 saniyede bir, 10 kez rapor
vmstat 2 10

# Detaylı disk istatistikleri
vmstat -d

# Bellek istatistikleri (MB cinsinden)
vmstat -S M 1

# Disk I/O istatistikleri
vmstat -D

# Fork istatistikleri
vmstat -f
```

Çıktı Açıklaması

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 2  0      0 458256  84520 1234567   0    0     5    10  500 1000 15  5 78  2  0
```

Kolonlar:

- **r**: Run queue (çalışmayı bekleyen process sayısı)
- **b**: Uninterruptible sleep (I/O bekleyen process)
- **swpd**: Kullanılan swap miktarı
- **free**: Boş bellek
- **buff/cache**: Buffer ve cache belleği
- **si/so**: Swap in/out (KB/s)
- **bi/bo**: Block in/out (blocks/s)
- **in**: Interrupts per second
- **cs**: Context switches per second

- **us**: User CPU time
- **sy**: System CPU time
- **id**: Idle CPU time
- **wa**: I/O wait time
- **st**: Stolen time (virtualization)

Alarm Kriterleri

Metrik	Normal	Uyarı	Kritik
CPU Wait (wa)	<5%	5-15%	>15%
Run Queue (r)	<CPU sayısı	CPU sayısı x2	CPU sayısı x3
Swap In/Out (si/so)	0	>100 KB/s	>1 MB/s
Context Switches	<10000	10000-50000	>50000

SRE Best Practices

- Her 1-5 dakikada bir vmstat çalıştırarak baseline oluşturun
- Run queue sürekli CPU sayısından fazlaysa, CPU bottleneck vardır
- Swap kullanımı başladıysa bellek yetersizliği sinyalidir
- I/O wait sürekli yüksekse disk bottleneck araştırın

---

iostat - I/O Statistics

Genel Bakış

CPU kullanımı ve disk I/O performansını detaylı raporlar.

Kurulum

```
# Ubuntu/Debian
apt-get install sysstat

# RHEL/CentOS
yum install sysstat
```

Kullanım Örnekleri

```
# Temel kullanım - 2 saniyede bir
iostat 2

# Genişletilmiş disk istatistikleri
iostat -x 2

# Belirli disk için
```

```
iostat -x sda 2

# Human-readable format
iostat -xh 2

# CPU ve disk ayrıntılı
iostat -xc 2

# Kilobyte cinsinden
iostat -xk 2
```

Önemli Metrikler

```
iostat -x 2
```

```
Device  rrqm/s wrqm/s  r/s   w/s  rkB/s  wkB/s avgrq-sz avgqu-sz await r_await
w_await svctm %util
sda       0.00   5.00  10.0  20.0  400.0  800.0    80.00     0.50  16.7    5.0
22.0   2.5  75.0
```

Kritik Metrikler:

- **r/s, w/s**: Read/write requests per second
- **rkB/s, wkB/s**: Read/write kilobytes per second
- **await**: Average wait time (ms) - I/O queue + service time
- **r_await / w_await**: Read/write wait time
- **avgqu-sz**: Average queue size
- **%util**: Device utilization percentage
- **svctm**: Service time (deprecated, kullanmayın)

Alarm Kriterleri

Metrik	Normal	Uyarı	Kritik
%util	<70%	70-85%	>85%
await	<10ms (SSD), <20ms (HDD)	10-50ms (SSD), 20-100ms (HDD)	>50ms (SSD), >100ms (HDD)
avgqu-sz	<1	1-5	>5

SRE Best Practices

- **%util >90%** ve **await yüksek**: Disk bottleneck, RAID seviyesi artırın veya daha hızlı diskler kullanın
- **%util düşük** ama **await yüksek**: Yavaş diskler, SSD'ye geçiş düşünün
- **avgqu-sz sürekli yüksek**: I/O scheduler ayarlarını optimize edin

- NVMe diskler için await <1ms olmalı

---

htop - Interactive Process Viewer

Genel Bakış

`top` komutunun gelişmiş, renkli ve interaktif versiyonu.

Kurulum

```
# Ubuntu/Debian
apt-get install htop

# RHEL/CentOS
yum install htop
```

Kullanım Örnekleri

```
# Temel başlatma
htop

# Belirli user için
htop -u apache

# Tree view ile başlat
htop -t

# Delay ayarlama (1 saniye)
htop -d 10
```

Önemli Kısayollar

Tuş	Fonksiyon
F1	Yardım
F2	Setup menü
F3	Process arama
F4	Filtreleme
F5	Tree view
F6	Sıralama seçimi
F9	Kill process
F10	Çıkış

Tuş	Fonksiyon
Space	Process işaretle
U	User filtreleme
t	Tree view toggle
H	Thread göster/gizle
K	Thread gizle
Shift+P	CPU'ya göre sırala
Shift+M	Memory'ye göre sırala
Shift+T	Time'a göre sırala

CPU Bar Açıklaması

```
CPU[||||||||||||||||||||||||50.0%]
```

- **Yeşil**: User processes (normal priority)
- **Kırmızı**: Kernel processes
- **Mavi**: Low priority processes
- **Turuncu**: IRQ time
- **Mor**: Soft IRQ time
- **Gri**: I/O wait
- **Açık Mavi**: Virtualization (steal time)

Memory Bar Açıklaması

```
Mem[||||||||||||||||||2.5G/16.0G]
```

- **Yeşil**: Used memory
- **Mavi**: Buffer memory
- **Turuncu**: Cache memory

Alarm Kriterleri

Metrik	Normal	Uyarı	Kritik
CPU Usage	<70%	70-85%	>85%
Memory Usage	<80%	80-90%	>90%
Load Average	<CPU count	CPU count x1.5	>CPU count x2
Swap Usage	0	>10%	>25%

SRE Best Practices

- Load average'ı izleyin: 1min, 5min, 15min değerlerini karşılaştırın
- Tree view ile parent-child process ilişkilerini görün
- Memory leak şüphesi varsa TIME+ kolonuna bakın (sürekli artan processleri izleyin)
- Zombie process'leri tespit edin (Z status)
- Nice değerleri ayarlayarak process önceliklerini yönetin

---

dstat - Versatile Resource Statistics

Genel Bakış

vmstat, iostat, netstat ve ifstat'ın yeteneklerini birleştiren çok yönlü araç.

Kurulum

```
# Ubuntu/Debian
apt-get install dstat

# RHEL/CentOS
yum install dstat
```

Kullanım Örnekleri

```
# Temel kullanım (default: cpu, disk, net, paging, system)
dstat

# Custom display
dstat -cdngy

# İlk 10 CPU tüketen process
dstat --top-cpu

# İlk 10 bellek tüketen process
dstat --top-mem

# Disk I/O detaylı
dstat -d -D sda,sdb

# Network detaylı
dstat -n -N eth0,eth1

# Tam paket
dstat -tcmsdn --top-cpu --top-mem

# CSV export
dstat --output /var/log/dstat.csv 5
```

Popüler Kombinasyonlar

```
# Web server monitoring
dstat -tclmdrn --tcp --udp

# Database server monitoring
dstat -tclmdr --disk-util --io --top-io

# High-frequency trading / Low latency
dstat -tcmsdn --socket --tcp --top-latency

# Full system overview
dstat -taf --top-cpu --top-mem --top-io-adv
```

Plugin Listesi

```
# Tüm pluginleri listele
dstat --list

# Önemli pluginler:
# --cpu-adv: L1/L2 cache misses
# --mem-adv: Inactive/active memory
# --disk-util: Disk utilization
# --tcp: TCP statistics
# --udp: UDP statistics
# --socket: Socket statistics
# --top-bio: Top block I/O processes
```

Alarm Kriterleri

dstat çıktısı vmstat ve iostat'a benzer kriterlere sahiptir, ancak daha kapsamlıdır.

SRE Best Practices

- Cron job ile periyodik loglar toplayın: `dstat --output /var/log/dstat-$(date +%Y%m%d).csv 60 > /dev/null 2>&1`
- Network bottleneck için `--net` ve `--tcp` kullanın
- Disk performans sorunları için `--disk-util --io --top-io` kombinasyonu
- Real-time monitoring için `watch -n1 dstat -c -m -d -n --top-cpu`

---

sar - System Activity Reporter

Genel Bakış

Sistem performans verilerini toplayan, kaydeden ve raporlayan en kapsamlı araç. Tarihsel veri analizi için vazgeçilmez.

Kurulum ve Aktivasyon

```
# Ubuntu/Debian
apt-get install sysstat
systemctl enable sysstat
systemctl start sysstat

# RHEL/CentOS
yum install sysstat
systemctl enable sysstat
systemctl start sysstat

# Config dosyası: /etc/default/sysstat veya /etc/sysconfig/sysstat
# ENABLED="true" yapın
```

Veri Toplama Ayarları

```
# /etc/cron.d/sysstat dosyasını düzenleyin
# Örnek: Her 5 dakikada bir veri topla
*/5 * * * * root /usr/lib64/sa/sa1 1 1

# Her gün gece rapor oluştur
59 23 * * * root /usr/lib64/sa/sa2 -A
```

Kullanım Örnekleri

```
# CPU kullanımı (tüm CPU'lar)
sar -u 2 10

# CPU kullanımı (her core ayrı)
sar -P ALL 2 10

# Memory kullanımı
sar -r 2 10

# Memory statistics (detaylı)
sar -R 2 10

# Swap kullanımı
sar -S 2 10

# I/O ve transfer rate
sar -b 2 10

# Disk I/O (her disk için)
sar -d 2 10

# Network statistics
```

```
sar -n DEV 2 10

# TCP statistics
sar -n TCP 2 10

# Load average ve task statistics
sar -q 2 10

# Paging statistics
sar -B 2 10

# Context switches ve interrupts
sar -w 2 10
```

Tarihsel Veri Analizi

```
# Bugünün verisi
sar -u

# Belirli tarih (format: DD)
sar -u -f /var/log/sa/sa15

# Belirli saat aralığı
sar -u -s 09:00:00 -e 17:00:00

# Dünün verisi
sar -u -f /var/log/sa/sa$(date -d yesterday +%d)

# Tüm metrikleri göster
sar -A

# Son 3 saatin CPU kullanımı
sar -u -s $(date -d '3 hours ago' +%H:%M:%S)
```

Performans Analizi Senaryoları

1. CPU Bottleneck Analizi

```
# CPU kullanımı ve queue
sar -u -q 1 10

# İdeal çıktı:
# %idle > 20
# runq-sz < CPU count
# %iowait < 5
```

2. Memory Bottleneck Analizi

```
# Memory ve swap
sar -r -S 1 10

# İdeal çıktı:
# %memused < 90
# kbmemfree > 500MB
# %swpused = 0
```

3. Disk Bottleneck Analizi

```
# Disk I/O
sar -d -p 1 10

# İdeal çıktı:
# %util < 80
# await < 20ms (HDD), <10ms (SSD)
# avgqu-sz < 2
```

4. Network Bottleneck Analizi

```
# Network interface
sar -n DEV 1 10

# TCP retransmissions
sar -n TCP 1 10

# İdeal çıktı:
# rxkB/s ve txkB/s < interface capacity
# retrans/s < 1% of total packets
```

Rapor Oluşturma

```
# Günlük CPU raporu HTML formatında
sadf -g /var/log/sa/sa$(date +%d) -- -u > cpu_report.svg

# CSV export
sadf -d /var/log/sa/sa$(date +%d) -- -u > cpu_report.csv

# JSON format
sadf -j /var/log/sa/sa$(date +%d) -- -u > cpu_report.json

# XML format
sadf -x /var/log/sa/sa$(date +%d) -- -u > cpu_report.xml
```

Alarm Kriterleri

Metrik	Komut	Normal	Uyarı	Kritik
CPU %idle	sar -u	>30%	10-30%	<10%
CPU %iowait	sar -u	<5%	5-15%	>15%
Memory %memused	sar -r	<80%	80-90%	>90%
Swap %swpused	sar -S	0%	1-10%	>10%
Load Average	sar -q	<CPU count	1-2x CPU	>2x CPU
Disk %util	sar -d	<70%	70-85%	>85%
Network %ifutil	sar -n DEV	<70%	70-85%	>85%

SRE Best Practices

- **Tarihsel analiz**: sar verilerini en az 30 gün saklayın
- **Retention policy**: `/etc/sysstat/sysstat` dosyasında `HISTORY=30` ayarlayın
- **Baseline oluşturma**: Haftalık raporlar oluşturup normal değerleri belirleyin
- **Incident response**: Sorun anında son 24 saatin sar verilerini toplayın
- **Capacity planning**: Aylık trendleri analiz edin
- **Automation**: sar verileri üzerinde script'ler çalıştırarak otomatik alarm oluşturun

```
# Örnek monitoring scripti
#!/bin/bash
CPU_IDLE=$(sar -u 1 1 | grep Average | awk '{print $NF}')
if (( $(echo "$CPU_IDLE < 10" | bc -l) )); then
    echo "CRITICAL: CPU idle is ${CPU_IDLE}%"
    # Send alert
fi
```

---

iotop - I/O Monitor

Genel Bakış

Hangi processlerin ne kadar disk I/O yaptığını real-time gösteren htop'un I/O versiyonu.

Kurulum

```
# Ubuntu/Debian
apt-get install iotop

# RHEL/CentOS
yum install iotop
```

Kullanım Örnekleri

```
# Temel kullanım (root gerekli)
sudo iotop

# Sadece aktif I/O yapan processleri göster
sudo iotop -o

# Batch mode (non-interactive)
sudo iotop -b -n 5

# Process yerine thread göster
sudo iotop -P

# Accumulated I/O göster
sudo iotop -a

# Specific delay
sudo iotop -d 2

# Quiet mode (sadece total I/O)
sudo iotop -q
```

Çıktı Açıklaması

```
Total DISK READ:       50.00 M/s | Total DISK WRITE:      100.00 M/s
Current DISK READ:     45.00 M/s | Current DISK WRITE:     95.00 M/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 1234 be/4 mysql      10.00 M/s   50.00 M/s  0.00 %  25.00 % mysqld
 5678 be/4 postgres    5.00 M/s   20.00 M/s  0.00 %  10.00 % postgres
```

Kolonlar:

- **TID**: Thread ID (veya PID)
- **PRIO**: I/O priority (be=best effort, rt=real-time, idle=idle)
- **DISK READ/WRITE**: Current I/O rate
- **SWAPIN**: Swap in percentage
- **IO>**: I/O wait percentage
- **COMMAND**: Process/thread name

Interaktif Kısayollar

Tuş	Fonksiyon
o	Sadece I/O yapan processleri göster
p	Process/thread toggle

Tuş	Fonksiyon
a	Accumulated I/O toggle
q	Çıkış
r	Reverse sorting
→ / ←	Sıralama kolonu değiştir

Alarm Kriterleri

Metrik	Normal	Uyarı	Kritik
Total Disk I/O	<50 MB/s	50-100 MB/s	>100 MB/s (SATA SSD max ~550 MB/s)
Single Process I/O	<10 MB/s	10-50 MB/s	>50 MB/s
I/O Wait %	<5%	5-15%	>15%

SRE Best Practices

- **I/O profiling**: Hangi processlerin en çok I/O yaptığını belirleyin
- **Runaway process**: Beklenmedik yüksek I/O yapan processi tespit edin
- **Database optimization**: Database server'da hangi sorguların I/O yaptığını görün
- **Backup impact**: Backup'ların sistem üzerindeki I/O etkisini ölçün
- **Log analysis**: Log yazma işlemlerinin I/O overhead'ini belirleyin

```
# Top 5 I/O consumer'ları batch modda kaydet
sudo iotop -b -n 100 -d 5 -o > /var/log/iotop-$(date +%Y%m%d-%H%M).log
```

---

atop - Advanced System Monitor

Genel Bakış

En kapsamlı sistem performans monitörü. CPU, memory, disk, network ve process level detayları tek araçta toplar. Tarihsel veri saklama özelliği vardır.

Kurulum

```
# Ubuntu/Debian
apt-get install atop

# RHEL/CentOS
yum install atop

# Service'i etkinleştir (her 10 dakikada snapshot alır)
systemctl enable atop
systemctl start atop
```

Kullanım Örnekleri

```
# Temel kullanım
atop

# 5 saniyede bir güncelle
atop 5

# Belirli tarihten log oku
atop -r /var/log/atop/atop_20241016

# Belirli saat aralığı
atop -r /var/log/atop/atop_20241016 -b 09:00 -e 17:00

# Sadece CPU kullanımı yüksek olanları göster
atop -c

# Sadece memory kullanımı yüksek olanları göster
atop -m

# Sadece disk I/O yapanları göster
atop -d

# Network kullanımı
atop -n

# Process accounting ile detaylı tracking
atop -w /tmp/atop.log 5
```

Ana Ekran Görünümü

atop ekranı şu bölümlere ayrılır:

1. **PRC**: Process seviyesi statistics
2. **CPU**: CPU utilization
3. **CPL**: CPU load ve queue
4. **MEM**: Memory usage
5. **SWP**: Swap usage
6. **PAG**: Paging statistics
7. **LVM/MDD**: Logical volumes
8. **DSK**: Disk utilization
9. **NET**: Network utilization

Interaktif Kısayollar

Tuş	Fonksiyon
g	Generic output (default)

Tuş	Fonksiyon
m	Memory sorting
c	CPU sorting
d	Disk sorting
n	Network sorting
a	Automatic sorting
C	Command line toggle
u	User filter
p	Process filter
t	Forward in time (log mode)
T	Backward in time (log mode)
b	Jump to time (log mode)
r	Reset interval
v	Version info
z	Pause
q	Quit

Kritik Metrikler

```
PRC | sys  0.50s | user 2.30s | #proc  234 | #tslpu   8 | #tslpi   2 | #zombie  0
| #exit    0 |
CPU | sys      5% | user    23% | irq      1% | idle  270% | wait    1% | curf
2.4GHz | curscal ?% |
CPL | avg1  1.25 | avg5  1.50 | avg15  1.35 | csw   5234 | intr  8392 | numcpu   4
|
MEM | tot  16.0G | free  4.2G | cache 8.5G | buff  256M | slab  1.2G | shrss 120M
| vmcom 10.5G | vmlim 8.0G |
SWP | tot   8.0G | free  7.5G |  | vmcom 10.5G | vmlim 24.0G |
PAG | scan     0 | stall    0 | swin     0 | swout    0 |
DSK |  sda | busy  35% | read  128 | write  256 | MBr/s   5.2 | MBw/s  10.8 | avio
4.2ms |
NET | eth0 | pcki  1250 | pcko  980 | si 1500 Kbps | so 1200 Kbps | erri  0 | erro
0 |
```

Process-Level Detaylar

```
  PID  SYSCPU  USRCPU  VGROW   RGROW  RDDSK  WRDSK   ST   EXC  S  CPU  CMD
 1234    0.15    0.45  125M    80M   5.0M   10.0M  --    0   R   6%  mysqld
```

```
  5678    0.08    0.22   50M    30M   2.0M    4.0M  --    0   S   3%  nginx
```

Tarihsel Analizi

```
# Dünün 14:00-15:00 arası CPU kullanımı
atop -r /var/log/atop/atop_$(date -d yesterday +%Y%m%d) -b 14:00 -e 15:00 -c

# Son 7 günün maksimum CPU kullanımını bul
for i in {0..7}; do
  date=$(date -d "$i days ago" +%Y%m%d)
  echo "=== $date ==="
  atop -r /var/log/atop/atop_$date -PCPU | head -20
done

# Specific process tracking
atop -r /var/log/atop/atop_20241016 | grep mysql
```

atopsar - sar benzeri raporlama

```
# CPU report
atopsar -c -r /var/log/atop/atop_20241016

# Memory report
atopsar -m -r /var/log/atop/atop_20241016

# Disk report
atopsar -d -r /var/log/atop/atop_20241016

# Network report
atopsar -n -r /var/log/atop/atop_20241016
```

Alarm Kriterleri

Metrik	Normal	Uyarı	Kritik
CPU %idle	>30%	10-30%	<10%
Load Average	<CPU count	1-2x CPU	>2x CPU
Memory Free	>20%	10-20%	<10%
Swap Usage	0%	1-10%	>10%
Disk Busy	<70%	70-85%	>85%
Zombie Process	0	1-5	>5
Exit Count	<10	10-50	>50 (abnormal)

SRE Best Practices

1. Retention Configuration

```
# /etc/default/atop dosyasını düzenle
LOGINTERVAL=600        # 10 dakika
LOGGENERATIONS=28      # 28 gün sakla
LOGPATH=/var/log/atop
```

2. Critical Process Monitoring

```
# MySQL monitoring
atop -g mysql 5

# High-memory process alert
atop -m 5 | awk '$5 > 1000000 {print $0}'  # 1GB üzeri
```

3. Performance Investigation Workflow

```
# 1. Genel duruma bak
atop

# 2. CPU bottleneck varsa
atop -c

# 3. Memory bottleneck varsa
atop -m

# 4. Disk bottleneck varsa
atop -d

# 5. Network bottleneck varsa
atop -n
```

4. Automated Reporting

```
#!/bin/bash
# /usr/local/bin/daily-atop-report.sh

DATE=$(date -d yesterday +%Y%m%d)
REPORT_DIR="/var/reports/atop"

mkdir -p $REPORT_DIR
```

```
# CPU Report
atopsar -c -r /var/log/atop/atop_$DATE > $REPORT_DIR/cpu_$DATE.txt

# Memory Report
atopsar -m -r /var/log/atop/atop_$DATE > $REPORT_DIR/mem_$DATE.txt

# Disk Report
atopsar -d -r /var/log/atop/atop_$DATE > $REPORT_DIR/disk_$DATE.txt

# Top CPU consumers
atop -r /var/log/atop/atop_$DATE -PCPU | head -50 > $REPORT_DIR/top_cpu_$DATE.txt

# Top Memory consumers
atop -r /var/log/atop/atop_$DATE -PMEM | head -50 > $REPORT_DIR/top_mem_$DATE.txt
```

---

perf - Performance Analysis

Genel Bakış

Linux kernel'in performance counters framework'ü. CPU performance counters, tracepoints ve dynamic probes kullanarak detaylı analiz yapar.

Kurulum

```
# Ubuntu/Debian
apt-get install linux-tools-common linux-tools-generic linux-tools-$(uname -r)

# RHEL/CentOS
yum install perf

# Verify
perf --version
```

Temel Kullanım Örnekleri

1. System-Wide Profiling

```
# Tüm sistem CPU profiling (5 saniye)
perf record -a -g sleep 5

# Profiling sonucunu görüntüle
perf report

# Interactive TUI
perf report -i perf.data

# Call graph ile
```

```
perf record -a -g -F 99 sleep 10
perf report --stdio
```

2. Specific Process Profiling

```
# PID ile process profiling
perf record -p <PID> -g sleep 10

# Command profiling
perf record -g ./my_application

# Multi-threaded application
perf record -g --call-graph dwarf ./my_app
```

3. CPU Events

```
# Tüm CPU events listesi
perf list

# Specific events tracking
perf stat -e cycles,instructions,cache-references,cache-misses ./my_app

# Branch prediction misses
perf stat -e branches,branch-misses ./my_app

# L1/L2/L3 cache misses
perf stat -e L1-dcache-loads,L1-dcache-load-misses,LLC-loads,LLC-load-misses
./my_app
```

4. Real-time Monitoring

```
# Top-like interface
perf top

# Specific function
perf top -G

# Kernel functions
perf top -K

# User-space only
perf top -U

# System-wide with call graph
perf top -g
```

5. CPU Flame Graphs

```
# Record data
perf record -F 99 -a -g -- sleep 60

# Generate FlameGraph (requires FlameGraph toolkit)
perf script | stackcollapse-perf.pl | flamegraph.pl > flamegraph.svg

# FlameGraph toolkit: https://github.com/brendangregg/FlameGraph
```

Advanced Use Cases

1. CPU Cycle Analysis

```
# Detailed CPU statistics
perf stat -d ./my_application

# Very detailed (includes L3 cache)
perf stat -d -d ./my_application

# All available metrics
perf stat -a -d -d -d sleep 10
```

2. Context Switch Analysis

```
# Record context switches
perf record -e sched:sched_switch -a sleep 10

# Report context switches
perf script

# Count by process
perf script | awk '{print $1}' | sort | uniq -c | sort -rn | head -20
```

3. Page Fault Analysis

```
# Record page faults
perf record -e page-faults -a -g sleep 10

# Major page faults (disk I/O required)
perf record -e major-faults -a -g sleep 10
```

```
# Minor page faults (memory only)
perf record -e minor-faults -a -g sleep 10
```

4. Lock Contention Analysis

```
# Record lock events
perf record -e lock:* -a -g sleep 10

# Analyze lock contention
perf lock record -a sleep 10
perf lock report
```

5. Memory Access Patterns

```
# Record memory loads/stores
perf mem record -a sleep 10

# Report memory access patterns
perf mem report

# TLB misses
perf stat -e dTLB-loads,dTLB-load-misses,iTLB-loads,iTLB-load-misses sleep 10
```

Performance Tuning Scenarios

Scenario 1: High CPU Usage Investigation

```
# 1. Identify hot functions
perf record -a -g sleep 30
perf report --sort comm,dso,symbol

# 2. Check IPC (Instructions Per Cycle)
perf stat -e cycles,instructions ./app
# IPC > 1.0 is good, < 0.5 indicates stalls

# 3. Check branch prediction
perf stat -e branches,branch-misses ./app
# branch-miss rate < 5% is acceptable
```

Scenario 2: Cache Miss Investigation

```
# L1 cache analysis
perf stat -e L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores,L1-dcache-
```

```
store-misses ./app

# L2 cache analysis
perf stat -e l2_rqsts.all_demand_data_rd,l2_rqsts.demand_data_rd_miss ./app

# LLC (Last Level Cache) analysis
perf stat -e LLC-loads,LLC-load-misses,LLC-stores,LLC-store-misses ./app

# Cache miss rate < 3% is good, > 10% needs optimization
```

Scenario 3: Function-Level Profiling

```
# Annotate specific function
perf record -g ./app
perf annotate <function_name>

# Shows assembly code with performance counters
# Helps identify hot loops and optimization opportunities
```

perf trace - System Call Tracing

```
# Trace all system calls (like strace but faster)
perf trace

# Trace specific process
perf trace -p <PID>

# Trace specific syscalls
perf trace -e open,close,read,write

# Show duration and summary
perf trace -s

# With timestamp
perf trace --duration 100  # Only syscalls taking >100ms
```

Alarm Kriterleri

Metrik	Normal	Uyarı	Kritik
IPC (Instructions/Cycle)	>1.0	0.5-1.0	<0.5
Branch Miss Rate	<5%	5-10%	>10%
L1 Cache Miss Rate	❤️%	3-5%	>5%
LLC Miss Rate	<1%	1-5%	>5%

Metrik	Normal	Uyarı	Kritik
Page Faults/sec	<100	100-1000	>1000
Context Switches/sec	<5000	5000-15000	>15000

SRE Best Practices

1. Baseline Performance

```
# Create performance baseline
perf stat -d -d -d -r 10 ./application > baseline.txt 2>&1

# Compare against baseline regularly
perf stat -d -d -d -r 10 ./application > current.txt 2>&1
diff baseline.txt current.txt
```

2. Production Profiling

```
# Low overhead production profiling (99 Hz sampling)
perf record -F 99 -a -g -o /tmp/perf.data sleep 60

# Minimal impact: -F 49 (49 Hz)
perf record -F 49 -a -g -o /tmp/perf.data sleep 300
```

3. Automated Performance Regression Detection

```
#!/bin/bash
# /usr/local/bin/perf-regression-check.sh

APP="./my_application"
BASELINE_IPC=1.2
BASELINE_CACHE_MISS=2.5

CURRENT=$(perf stat -e cycles,instructions $APP 2>&1 | grep "insn per cycle")
IPC=$(echo $CURRENT | awk '{print $4}')

if (( $(echo "$IPC < $BASELINE_IPC * 0.9" | bc -l) )); then
    echo "ALERT: IPC regression detected. Current: $IPC, Baseline: $BASELINE_IPC"
    # Send alert
fi
```

4. Kernel Symbol Resolution

```
# Install debug symbols
apt-get install linux-image-$(uname -r)-dbgsym  # Ubuntu
debuginfo-install kernel  # RHEL

# Or use kallsyms
cat /proc/kallsyms > /tmp/kallsyms.txt
```

5. Continuous Profiling

```
# Cron job for daily profiling
0 2 * * * perf record -F 99 -a -g -o /var/log/perf/perf-$(date +\%Y\%m\%d).data
sleep 300
```

---

bpftrace / eBPF - Dynamic Tracing

Genel Bakış

eBPF (extended Berkeley Packet Filter) modern Linux kernelin en güçlü observability teknolojisi. Kernel'e dinamik olarak safe code inject ederek zero-overhead tracing sağlar.

Kurulum

```
# Ubuntu/Debian 20.04+
apt-get install bpftrace bpfcc-tools linux-headers-$(uname -r)

# RHEL/CentOS 8+
yum install bpftrace bcc-tools kernel-devel

# Kernel version kontrolü (minimum 4.9, ideal 5.x+)
uname -r
```

bpftrace Kullanım Örnekleri

1. One-Liners - Quick Analysis

```
# CPU'da çalışan tüm processleri trace et
bpftrace -e 'profile:hz:99 { @[comm] = count(); }'

# Yeni process'leri takip et
bpftrace -e 'tracepoint:sched:sched_process_exec { printf("%s -> %s\n", comm,
str(args->filename)); }'

# Disk I/O latency histogramı
bpftrace -e 'tracepoint:block:block_rq_complete { @usecs = hist(args->nr_sector);
```

```
}'

# System call monitoring
bpftrace -e 'tracepoint:raw_syscalls:sys_enter { @[comm] = count(); }'

# TCP connection tracking
bpftrace -e 'kprobe:tcp_connect { printf("PID %d connecting\n", pid); }'

# File opens by process
bpftrace -e 'tracepoint:syscalls:sys_enter_openat { printf("%s opened %s\n", comm,
str(args->filename)); }'

# Memory allocation tracking
bpftrace -e 'tracepoint:kmem:kmalloc { @bytes = sum(args->bytes_alloc); }'
```

2. CPU Performance Analysis

```
# CPU profiling - hangi fonksiyonlar çalışıyor
bpftrace -e 'profile:hz:99 /pid == 1234/ { @[ustack] = count(); }'

# On-CPU time by process
bpftrace -e 'profile:hz:99 { @[comm, pid] = count(); }'

# Off-CPU time (blocked time)
cat > offcpu.bt << 'EOF'
#include <linux/sched.h>

kprobe:finish_task_switch
{
  $prev = (struct task_struct *)arg0;
  $nsecs = nsecs - @start[$prev->pid];

  if ($nsecs > 1000000) {
    @usecs[pid, comm] = hist($nsecs / 1000);
  }

  @start[pid] = nsecs;
}
EOF

bpftrace offcpu.bt
```

3. Memory Leak Detection

```
# Track malloc/free
cat > memleak.bt << 'EOF'
tracepoint:kmem:kmalloc
{
```

```
  @allocs[args->ptr] = args->bytes_alloc;
  @total = sum(args->bytes_alloc);
}

tracepoint:kmem:kfree
{
  if (@allocs[args->ptr]) {
    @total = @total - @allocs[args->ptr];
    delete(@allocs[args->ptr]);
  }
}

interval:s:10
{
  printf("Total allocated: %d bytes\n", @total);
  print(@total);
}
EOF

bpftrace memleak.bt
```

4. Disk I/O Analysis

```
# Block I/O latency by process
cat > biolatency.bt << 'EOF'
BEGIN
{
  printf("Tracing block I/O latency... Hit Ctrl-C to end.\n");
}

kprobe:blk_account_io_start
{
  @start[arg0] = nsecs;
}

kprobe:blk_account_io_done
/@start[arg0]/
{
  $latency = (nsecs - @start[arg0]) / 1000;
  @usecs = hist($latency);
  delete(@start[arg0]);
}

END
{
  clear(@start);
}
EOF

bpftrace biolatency.bt
```

5. Network Performance

```
# TCP retransmissions
bpftrace -e 'kprobe:tcp_retransmit_skb { @retrans[comm] = count(); }'

# TCP connection latency
cat > tcpconnlat.bt << 'EOF'
#include <net/sock.h>

kprobe:tcp_v4_connect
{
  @start[pid] = nsecs;
}

kretprobe:tcp_v4_connect
/@start[pid]/
{
  $latency = (nsecs - @start[pid]) / 1000000;
  printf("PID %d (%s) connected in %d ms\n", pid, comm, $latency);
  delete(@start[pid]);
}
EOF

bpftrace tcpconnlat.bt

# Network packet drops
bpftrace -e 'tracepoint:skb:kfree_skb { @[stack] = count(); }'
```

BCC Tools - Production-Ready Scripts

BCC (BPF Compiler Collection) hazır production-ready tools içerir:

```
# CPU profiling
profile-bpfcc 10    # 10 saniye CPU profile

# System call latency
syscount-bpfcc      # Syscall counts
funclatency-bpfcc vfs_read  # VFS read latency

# Block I/O
biolatency-bpfcc    # Block I/O latency histogram
biosnoop-bpfcc      # Live block I/O events
biotop-bpfcc        # Top block I/O by process

# Network
tcpconnect-bpfcc    # TCP active connections
tcpaccept-bpfcc     # TCP passive connections
tcpretrans-bpfcc    # TCP retransmissions
tcptop-bpfcc        # TCP throughput by host
```

```
# File System
opensnoop-bpfcc     # File opens
statsnoop-bpfcc     # stat() calls
syncsnoop-bpfcc     # sync() calls
ext4slower-bpfcc 10 # ext4 operations slower than 10ms

# Memory
memleak-bpfcc -p <PID>  # Memory leak detection
slabratetop-bpfcc       # Kernel slab allocator stats

# Process
execsnoop-bpfcc     # Process execution
exitsnoop-bpfcc     # Process exits
killsnoop-bpfcc     # Kill signals
```

Advanced Production Scenarios

Scenario 1: Database Query Latency

```
# MySQL query tracing
cat > mysql_queries.bt << 'EOF'
usdt:/usr/sbin/mysqld:mysql:query__start
{
  @query_start[tid] = nsecs;
  @query[tid] = str(arg0);
}

usdt:/usr/sbin/mysqld:mysql:query__done
/@query_start[tid]/
{
  $latency_ms = (nsecs - @query_start[tid]) / 1000000;

  if ($latency_ms > 100) {  // Slow queries > 100ms
    printf("SLOW QUERY (%d ms): %s\n", $latency_ms, @query[tid]);
  }

  @latency_hist = hist($latency_ms);
  delete(@query_start[tid]);
  delete(@query[tid]);
}
EOF

bpftrace mysql_queries.bt
```

Scenario 2: Application Function Tracing

```
# User-space function tracing (requires debug symbols)
bpftrace -e 'uprobe:/path/to/app:function_name { printf("Called with arg: %d\n",
```

```
arg0); }'

# Return value tracking
bpftrace -e 'uretprobe:/path/to/app:function_name { printf("Returned: %d\n",
retval); }'
```

Scenario 3: Container Monitoring

```
# Track which containers are doing I/O
cat > container_io.bt << 'EOF'
#include <linux/blkdev.h>

kprobe:blk_account_io_start
{
  $dev = ((struct request *)arg0)->rq_disk->disk_name;
  $cgroup = cgroup;
  @io[$cgroup, str($dev)] = count();
}

interval:s:5
{
  print(@io);
  clear(@io);
}
EOF

bpftrace container_io.bt
```

Security Monitoring with eBPF

```
# Detect privilege escalation attempts
bpftrace -e 'tracepoint:syscalls:sys_enter_setuid { if (args->uid == 0) {
printf("ALERT: %s (PID %d) attempting setuid(0)\n", comm, pid); } }'

# Monitor sensitive file access
bpftrace -e 'tracepoint:syscalls:sys_enter_openat /str(args->filename) ==
"/etc/shadow" || str(args->filename) == "/etc/passwd"/ { printf("ALERT: %s
accessing %s\n", comm, str(args->filename)); }'

# Track outbound connections
bpftrace -e 'kprobe:tcp_connect { printf("%s connecting to remote host\n", comm);
}'
```

Alarm Kriterleri

Metrik	Normal	Uyarı	Kritik

Metrik	Normal	Uyarı	Kritik
Syscall Latency	<100µs	100µs-1ms	>1ms
Block I/O Latency	<10ms	10-100ms	>100ms
TCP Retrans Rate	<0.1%	0.1-1%	>1%
Page Faults	<1000/s	1000-5000/s	>5000/s
Context Switches	<10000/s	10k-50k/s	>50k/s

SRE Best Practices

1. Safety First

```
# Her zaman timeout kullan
timeout 60 bpftrace script.bt

# Production'da sampling rate düşük tutun
bpftrace -e 'profile:hz:49 { ... }'  # 99 yerine 49 Hz

# Resource limit check
ulimit -l unlimited  # BPF map size için gerekli
```

2. Performance Overhead

- **bpftrace/eBPF overhead**: Genelde <1% CPU
- **Yüksek frekanslı events**: Overhead artabilir (>10%)
- **Production recommendation**: Hz:49 veya daha düşük sampling

3. Debugging Workflow

```
# 1. Identify problem with standard tools
vmstat, iostat, htop

# 2. Deep dive with eBPF
bpftrace one-liners

# 3. Custom tracing script
Geliştir ve test et

# 4. Production deployment
Monitoring sistemine entegre et
```

4. Integration with Monitoring

```
# Export metrics to Prometheus
cat > export_metrics.sh << 'EOF'
#!/bin/bash
while true; do
  bpftrace -e 'profile:hz:99 { @cpu[comm] = count(); } interval:s:10 {
print(@cpu); clear(@cpu); exit(); }' \
  | grep -v "^@" \
  | awk '{print "cpu_usage{process=\""$1"\"} "$2}' \
  > /var/lib/node_exporter/textfile_collector/bpf_cpu.prom

  sleep 10
done
EOF
```

---

glances - Cross-platform Monitoring

Genel Bakış

Python ile yazılmış, modern ve kullanıcı dostu sistem monitoring tool. Web UI, API ve alerting özellikleri ile SRE ekipleri için ideal.

Kurulum

```
# Ubuntu/Debian
apt-get install glances

# RHEL/CentOS (EPEL repository gerekli)
yum install epel-release
yum install glances

# pip ile (güncel versiyon)
pip3 install glances

# Docker ile
docker run -it --rm --pid host --network host -v
/var/run/docker.sock:/var/run/docker.sock:ro nicolargo/glances
```

Kullanım Örnekleri

1. Temel Kullanım

```
# Standart başlatma
glances

# Refresh interval (saniye)
glances -t 2
```

```
# Minimal mode (sadece CPU, RAM, SWAP)
glances --disable-process

# Process tree
glances --tree

# Per-CPU stats
glances --percpu

# Show network cumulative
glances --network-cumul
```

2. Web Server Mode

```
# Web sunucusu olarak başlat (port 61208)
glances -w

# Custom port
glances -w --port 8080

# Password protection
glances -w --password

# Access: http://localhost:61208
```

3. Client/Server Mode

```
# Server tarafı
glances -s

# Client tarafı
glances -c <server_ip>

# Custom port
glances -s -B 0.0.0.0 -p 61209
glances -c <server_ip> -p 61209

# Password protected
glances -s --password
glances -c <server_ip> --password
```

4. Export Options

```
# CSV export
glances --export csv --export-csv-file /var/log/glances.csv
```

```
# InfluxDB export
glances --export influxdb --influxdb-host localhost --influxdb-port 8086

# Prometheus export
glances --export prometheus

# Grafana/InfluxDB stack
glances --export influxdb2

# JSON export
glances --export-json /var/log/glances.json

# RESTful API
glances -w --disable-webui  # Sadece API
# Access: http://localhost:61208/api/3/cpu
```

5. Docker Monitoring

```
# Docker container monitoring
glances --docker

# Kubernetes pod monitoring (experimental)
glances --kubernetes
```

Interaktif Kısayollar

Tuş	Fonksiyon
h	Yardım
q	Çıkış
a	Sort automatically
c	Sort by CPU
m	Sort by MEM
p	Sort by process name
i	Sort by I/O
t	Sort by TIME+
d	Show/hide disk I/O
f	Show/hide filesystem
n	Show/hide network
s	Show/hide sensors

Tuş	Fonksiyon
w	Delete warning logs
x	Delete critical logs
1	Global CPU / Per CPU toggle
I	Show/hide IP address
D	Show/hide Docker
u	View cumulative network
/	Process filter
E	Erase process filter

Advanced Configuration

Configuration File

```
# Configuration dosyası: ~/.config/glances/glances.conf
# veya /etc/glances/glances.conf

cat > ~/.config/glances/glances.conf << 'EOF'
[global]
# Refresh interval in seconds
refresh=2
# HDD temperature warning threshold
temperature_hdd_careful=45
temperature_hdd_warning=52
temperature_hdd_critical=60

[cpu]
# CPU thresholds
user_careful=50
user_warning=70
user_critical=90
system_careful=50
system_warning=75
system_critical=90
iowait_careful=40
iowait_warning=60
iowait_critical=80

[mem]
# Memory thresholds (percentage)
careful=50
warning=70
critical=90

[memswap]
careful=50
```

```
warning=70
critical=90

[load]
# Load average thresholds (per CPU core)
careful=0.7
warning=1.0
critical=5.0

[network]
# Network thresholds (in Bytes/s)
rx_careful=40000000
rx_warning=60000000
rx_critical=80000000
tx_careful=10000000
tx_warning=25000000
tx_critical=50000000

[diskio]
# Disk I/O thresholds (in Bytes/s)
read_careful=10000000
read_warning=20000000
read_critical=30000000
write_careful=10000000
write_warning=20000000
write_critical=30000000

[fs]
# Filesystem thresholds
careful=50
warning=70
critical=90

[sensors]
# Temperature thresholds
temperature_core_careful=60
temperature_core_warning=70
temperature_core_critical=80

[processlist]
# Process list configuration
cpu_careful=50
cpu_warning=70
cpu_critical=90
mem_careful=50
mem_warning=70
mem_critical=90

[alerts]
# Alert configuration
disable=False
# Alert history size
max_events=20
EOF
```

Alert Actions

```
# Alert script configuration
cat >> ~/.config/glances/glances.conf << 'EOF'

[alert_command]
# Command to execute on alert
command=echo "ALERT: {0} on {1}" | mail -s "Glances Alert" admin@example.com
# {0} = alert message
# {1} = hostname
EOF
```

API Usage Examples

```
# Get all stats
curl http://localhost:61208/api/3/all

# CPU stats
curl http://localhost:61208/api/3/cpu

# Memory stats
curl http://localhost:61208/api/3/mem

# Load average
curl http://localhost:61208/api/3/load

# Network stats
curl http://localhost:61208/api/3/network

# Disk I/O
curl http://localhost:61208/api/3/diskio

# Process list (top 10 by CPU)
curl http://localhost:61208/api/3/processlist

# Docker containers
curl http://localhost:61208/api/3/docker

# Get limits (thresholds)
curl http://localhost:61208/api/3/cpu/limits

# Get specific process by name
curl http://localhost:61208/api/3/processlist/name/mysql
```

Prometheus Integration

```
# Start Glances with Prometheus exporter
glances --export prometheus --prometheus-port 9091

# Prometheus scrape config
cat >> /etc/prometheus/prometheus.yml << 'EOF'
scrape_configs:
  - job_name: 'glances'
    static_configs:
      - targets: ['localhost:9091']
        labels:
          instance: 'server1'
EOF

# Metrics available at: http://localhost:9091/metrics
```

Grafana Dashboard

```
# InfluxDB + Grafana Stack
# 1. Start Glances with InfluxDB export
glances --export influxdb2 \
  --influxdb2-host localhost \
  --influxdb2-port 8086 \
  --influxdb2-org myorg \
  --influxdb2-bucket glances \
  --influxdb2-token mytoken

# 2. Grafana'da InfluxDB data source ekle
# 3. Glances dashboard import et (ID: 2387)
```

Alarm Kriterleri

Metrik	Careful (Sarı)	Warning (Turuncu)	Critical (Kırmızı)
CPU Usage	50%	70%	90%
Memory Usage	50%	70%	90%
Swap Usage	50%	70%	90%
Load Average	0.7x CPU	1.0x CPU	5.0x CPU
Disk I/O Read	10 MB/s	20 MB/s	30 MB/s
Network RX	40 MB/s	60 MB/s	80 MB/s
Disk Usage	50%	70%	90%
Temperature	60°C	70°C	80°C

SRE Best Practices

1. Centralized Monitoring

```
# Multiple servers monitoring
# Server 1, 2, 3: glances -s -B 0.0.0.0
# Monitoring server:
cat > monitor.sh << 'EOF'
#!/bin/bash
tmux new-session -d -s glances
tmux split-window -h
tmux split-window -v

tmux send-keys -t 0 "glances -c server1" C-m
tmux send-keys -t 1 "glances -c server2" C-m
tmux send-keys -t 2 "glances -c server3" C-m

tmux attach -t glances
EOF
```

2. Automated Reporting

```
# Daily report script
cat > /usr/local/bin/glances-daily-report.sh << 'EOF'
#!/bin/bash
DATE=$(date +%Y-%m-%d)
REPORT_DIR="/var/reports/glances"

mkdir -p $REPORT_DIR

# Run glances for 1 hour, export to CSV
timeout 3600 glances --export csv --export-csv-file $REPORT_DIR/glances-$DATE.csv
-t 60

# Generate summary
cat > $REPORT_DIR/summary-$DATE.txt << EOL
Glances Daily Report - $DATE
================================

Max CPU Usage: $(awk -F',' 'NR>1 {if($2>max) max=$2} END {print max"%"}'
$REPORT_DIR/glances-$DATE.csv)
Max Memory Usage: $(awk -F',' 'NR>1 {if($3>max) max=$3} END {print max"%"}'
$REPORT_DIR/glances-$DATE.csv)
Max Load Average: $(awk -F',' 'NR>1 {if($4>max) max=$4} END {print max}'
$REPORT_DIR/glances-$DATE.csv)
EOL

# Email report
mail -s "Glances Daily Report - $DATE" admin@example.com <
$REPORT_DIR/summary-$DATE.txt
EOF
```

```
# Cron job
0 1 * * * /usr/local/bin/glances-daily-report.sh
```

3. Docker Container Monitoring

```
# Monitor all Docker containers
glances --docker --export influxdb2

# Alert on container issues
cat >> ~/.config/glances/glances.conf << 'EOF'
[docker]
# Docker thresholds
cpu_careful=50
cpu_warning=70
cpu_critical=90
mem_careful=70
mem_warning=80
mem_critical=90
EOF
```

4. REST API Integration

```
#!/usr/bin/env python3
# Glances API monitoring script

import requests
import json

GLANCES_URL = "http://localhost:61208/api/3"

def check_cpu():
    response = requests.get(f"{GLANCES_URL}/cpu")
    cpu = response.json()

    if cpu['total'] > 90:
        print(f"CRITICAL: CPU usage is {cpu['total']}%")
        # Send alert
    elif cpu['total'] > 70:
        print(f"WARNING: CPU usage is {cpu['total']}%")

def check_memory():
    response = requests.get(f"{GLANCES_URL}/mem")
    mem = response.json()

    percent = mem['percent']
    if percent > 90:
        print(f"CRITICAL: Memory usage is {percent}%")
    elif percent > 70:
```

```
        print(f"WARNING: Memory usage is {percent}%")

def check_disk():
    response = requests.get(f"{GLANCES_URL}/fs")
    filesystems = response.json()

    for fs in filesystems:
        percent = fs['percent']
        if percent > 90:
            print(f"CRITICAL: {fs['mnt_point']} is {percent}% full")

if __name__ == "__main__":
    check_cpu()
    check_memory()
    check_disk()
```

Alarm Kriterleri ve Eşik Değerleri

Genel Sistem Sağlığı Matrisi

Kategori	Metrik	İyi	Dikkat	Uyarı	Kritik	Acil
CPU	Usage	<60%	60-70%	70-85%	85-95%	>95%
	Load Avg	<CPU*0.7	CPU*0.7-1.0	CPU*1.0-1.5	CPU*1.5-2.0	>CPU*2.0
	I/O Wait	<5%	5-10%	10-20%	20-40%	>40%
	Context Switches	<10k/s	10-30k/s	30-50k/s	50-100k/s	>100k/s
Memory	Usage	<70%	70-80%	80-90%	90-95%	>95%
	Swap Usage	0%	1-5%	5-20%	20-50%	>50%
	Page Faults	<100/s	100-500/s	500-1k/s	1-5k/s	>5k/s
	OOM Kills	0	0	1/day	>1/day	>1/hour
Disk	Utilization	<60%	60-70%	70-85%	85-95%	>95%
	I/O Latency (SSD)	<5ms	5-10ms	10-25ms	25-50ms	>50ms
	I/O Latency (HDD)	<15ms	15-30ms	30-60ms	60-100ms	>100ms
	Queue Size	<1	1-2	2-5	5-10	>10
	IOPS (SSD)	<50k	50-80k	80-100k	100-150k	>150k
Network	Bandwidth Usage	<50%	50-70%	70-85%	85-95%	>95%
	Packet Loss	0%	0-0.1%	0.1-0.5%	0.5-1%	>1%
	Retransmissions	<0.1%	0.1-0.5%	0.5-1%	1-5%	>5%
	Connection Errors	0	<10/min	10-50/min	50-100/min	>100/min

Kategori	Metrik	İyi	Dikkat	Uyarı	Kritik	Acil
Process	Zombie Count	0	1-2	3-5	6-10	>10
	Thread Count	<500	500-1000	1000-2000	2000-5000	>5000
	File Descriptors	<50%	50-70%	70-85%	85-95%	>95%

Servis Tiplerine Göre Özel Eşikler

Web Server (Nginx/Apache)

```yaml
cpu_usage:
  warning: 70%
  critical: 85%
memory_usage:
  warning: 80%
  critical: 90%
connection_count:
  warning: 5000
  critical: 10000
request_rate:
  warning: 1000/s
  critical: 5000/s
response_time:
  warning: 500ms
  critical: 1000ms
```

Database Server (MySQL/PostgreSQL)

```yaml
cpu_usage:
  warning: 75%
  critical: 90%
memory_usage:
  warning: 85%
  critical: 92%
disk_io_wait:
  warning: 10%
  critical: 20%
connection_count:
  warning: 80% of max_connections
  critical: 90% of max_connections
slow_queries:
  warning: 10/min
  critical: 50/min
replication_lag:
  warning: 10s
  critical: 60s
```

Application Server (Java/Node.js)

```
cpu_usage:
  warning: 70%
  critical: 85%
memory_usage:
  warning: 80%
  critical: 90%
heap_usage:
  warning: 75%
  critical: 90%
gc_frequency:
  warning: 10/min
  critical: 30/min
gc_duration:
  warning: 100ms
  critical: 500ms
thread_count:
  warning: 200
  critical: 500
```

Alarm Response Matrix

Severity	Response Time	On-Call Action	Escalation
İyi	-	Monitoring	-
Dikkat	30 min	Log & Track	-
Uyarı	15 min	Investigate	After 30 min
Kritik	5 min	Immediate Action	After 15 min
Acil	Immediate	Emergency Response	Immediate Manager

Best Practices - SRE/DevOps Yaklaşımı

1. Monitoring Strategy

Layered Monitoring Approach

```
Layer 1: Real-time (1-5 sec intervals)
├── htop / glances
├── dstat
└── Quick health checks

Layer 2: Short-term (1-5 min intervals)
├── vmstat
├── iostat
├── sar
```

```
└── Baseline comparison

Layer 3: Historical (10-60 min intervals)
├── atop logs
├── sar archives
└── Trend analysis

Layer 4: Deep Dive (On-demand)
├── perf
├── bpftrace
└── Root cause analysis
```

Tool Selection Matrix

```
Quick Check      → htop, glances
CPU Analysis     → vmstat, sar -u, perf
Memory Analysis  → vmstat, sar -r, atop
Disk I/O         → iostat, iotop, sar -d
Network          → sar -n, dstat -n, bpftrace
Process Level    → htop, atop, iotop
Kernel Tracing   → perf, bpftrace
Historical       → sar, atop logs
Visualization    → glances web UI, Grafana
```

2. Data Collection and Retention

Sampling Intervals

```
# Real-time monitoring
vmstat 1    # CPU/Memory - 1 second
iostat -x 2 # Disk I/O - 2 seconds
dstat 1     # All stats - 1 second

# Regular monitoring
sar 5 720   # Every 5 min for 1 hour
atop 600    # Every 10 minutes

# Long-term monitoring
sar -o /var/log/sa/sa$(date +%d) 600 # 10 min intervals
```

Retention Policy

```
Real-time data:    Keep for 1 hour
Hourly data:       Keep for 7 days
Daily aggregates:  Keep for 30 days
```

```
Weekly aggregates: Keep for 6 months
Monthly aggregates: Keep for 2 years
```

Storage Requirements

```
# Örnek hesaplama (orta ölçekli sunucu)
sar logs:     ~50 MB/day × 30 days = 1.5 GB
atop logs:    ~100 MB/day × 30 days = 3 GB
perf data:    ~500 MB/capture (on-demand)
glances CSV:  ~10 MB/day × 30 days = 300 MB

Total:        ~5 GB/month/server
```

3. Automation Scripts

Health Check Script

```
#!/bin/bash
# /usr/local/bin/system-health-check.sh

LOG_FILE="/var/log/health-check.log"
ALERT_EMAIL="devops@example.com"

log() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a $LOG_FILE
}

check_cpu() {
    CPU_USAGE=$(vmstat 1 2 | tail -1 | awk '{print 100-$15}')
    if [ $CPU_USAGE -gt 90 ]; then
        log "CRITICAL: CPU usage is ${CPU_USAGE}%"
        echo "CPU usage critical: ${CPU_USAGE}%" | mail -s "Alert: High CPU"
$ALERT_EMAIL
    elif [ $CPU_USAGE -gt 70 ]; then
        log "WARNING: CPU usage is ${CPU_USAGE}%"
    fi
}

check_memory() {
    MEM_USAGE=$(free | grep Mem | awk '{print int($3/$2 * 100)}')
    SWAP_USAGE=$(free | grep Swap | awk '{print int($3/$2 * 100)}')

    if [ $MEM_USAGE -gt 90 ]; then
        log "CRITICAL: Memory usage is ${MEM_USAGE}%"
        echo "Memory usage critical: ${MEM_USAGE}%" | mail -s "Alert: High Memory"
$ALERT_EMAIL
    fi
```

```
    if [ $SWAP_USAGE -gt 50 ]; then
        log "CRITICAL: Swap usage is ${SWAP_USAGE}%"
        echo "Swap usage critical: ${SWAP_USAGE}%" | mail -s "Alert: High Swap"
$ALERT_EMAIL
    fi
}

check_disk() {
    df -h | grep -vE '^Filesystem|tmpfs|cdrom' | awk '{print $5 " " $6}' | while
read usage mount; do
        usage_num=${usage%\%}
        if [ $usage_num -gt 90 ]; then
            log "CRITICAL: Disk usage on $mount is $usage"
            echo "Disk $mount is ${usage} full" | mail -s "Alert: Disk Full"
$ALERT_EMAIL
        elif [ $usage_num -gt 80 ]; then
            log "WARNING: Disk usage on $mount is $usage"
        fi
    done
}

check_load() {
    LOAD_5MIN=$(uptime | awk -F'load average:' '{print $2}' | awk -F',' '{print
$2}' | xargs)
    CPU_COUNT=$(nproc)
    LOAD_THRESHOLD=$(echo "$CPU_COUNT * 2" | bc)

    if (( $(echo "$LOAD_5MIN > $LOAD_THRESHOLD" | bc -l) )); then
        log "CRITICAL: Load average is $LOAD_5MIN (threshold: $LOAD_THRESHOLD)"
        echo "Load average critical: $LOAD_5MIN" | mail -s "Alert: High Load"
$ALERT_EMAIL
    fi
}

check_services() {
    SERVICES=("nginx" "mysql" "redis")

    for service in "${SERVICES[@]}"; do
        if ! systemctl is-active --quiet $service; then
            log "CRITICAL: Service $service is down"
            echo "Service $service is down" | mail -s "Alert: Service Down"
$ALERT_EMAIL
        fi
    done
}

main() {
    log "=== Starting Health Check ==="
    check_cpu
    check_memory
    check_disk
    check_load
    check_services
    log "=== Health Check Complete ==="
```

```
}

main
```

Performance Baseline Script

```
#!/bin/bash
# /usr/local/bin/create-baseline.sh

BASELINE_DIR="/var/baseline"
DATE=$(date +%Y%m%d-%H%M%S)

mkdir -p $BASELINE_DIR

echo "Creating performance baseline at $DATE"

# CPU baseline
echo "=== CPU Baseline ===" > $BASELINE_DIR/baseline-$DATE.txt
vmstat 1 60 >> $BASELINE_DIR/baseline-$DATE.txt
sar -u 1 60 >> $BASELINE_DIR/baseline-$DATE.txt

# Memory baseline
echo "=== Memory Baseline ===" >> $BASELINE_DIR/baseline-$DATE.txt
free -h >> $BASELINE_DIR/baseline-$DATE.txt
sar -r 1 60 >> $BASELINE_DIR/baseline-$DATE.txt

# Disk baseline
echo "=== Disk Baseline ===" >> $BASELINE_DIR/baseline-$DATE.txt
iostat -x 1 60 >> $BASELINE_DIR/baseline-$DATE.txt

# Network baseline
echo "=== Network Baseline ===" >> $BASELINE_DIR/baseline-$DATE.txt
sar -n DEV 1 60 >> $BASELINE_DIR/baseline-$DATE.txt

# Process baseline
echo "=== Process Baseline ===" >> $BASELINE_DIR/baseline-$DATE.txt
ps aux --sort=-%cpu | head -20 >> $BASELINE_DIR/baseline-$DATE.txt

echo "Baseline created: $BASELINE_DIR/baseline-$DATE.txt"
```

4. Incident Response Workflow

Performance Issue Investigation

```
# Step 1: Quick Overview (30 seconds)
glances -t 1        # Overall health
htop               # Process view
dstat -tcmsdn      # Real-time metrics
```

```
# Step 2: Identify Bottleneck (2 minutes)
vmstat 1 30        # CPU/Memory
iostat -x 1 30     # Disk I/O
sar -n DEV 1 30    # Network

# Step 3: Deep Dive (5-10 minutes)
# If CPU bottleneck:
perf top
perf record -a -g sleep 30
perf report

# If Memory bottleneck:
atop -m
cat /proc/meminfo
slabtop

# If Disk bottleneck:
iotop -o
perf record -e block:block_rq_complete -a sleep 30

# If Network bottleneck:
bpftrace -e 'kprobe:tcp_retransmit_skb { @[comm] = count(); }'
sar -n TCP,ETCP 1 30

# Step 4: Collect Evidence
tar czf evidence-$(date +%Y%m%d-%H%M%S).tar.gz \
    /var/log/sa/sa$(date +%d) \
    /var/log/atop/atop_$(date +%Y%m%d) \
    /var/log/syslog \
    /var/log/dmesg

# Step 5: Document
cat > incident-report-$(date +%Y%m%d-%H%M%S).md << EOF
# Incident Report

## Timeline
- Detection: $(date)
- Impact: [Describe]
- Duration: [Duration]

## Symptoms
[Output from monitoring tools]

## Root Cause
[Analysis]

## Resolution
[Actions taken]

## Prevention
[Future improvements]
EOF
```

5. Capacity Planning

Resource Trend Analysis

```
#!/bin/bash
# /usr/local/bin/capacity-report.sh

REPORT_FILE="/var/reports/capacity-$(date +%Y%m).txt"

cat > $REPORT_FILE << EOF
# Capacity Planning Report - $(date +%B\ %Y)

## CPU Trends (Last 30 days)
EOF

# CPU usage trend
echo "### Average CPU Usage by Day" >> $REPORT_FILE
for i in {1..30}; do
    date=$(date -d "$i days ago" +%Y%m%d)
    if [ -f /var/log/sa/sa$(date -d "$i days ago" +%d) ]; then
        avg=$(sar -u -f /var/log/sa/sa$(date -d "$i days ago" +%d) | grep Average
| awk '{print 100-$NF}')
        echo "$date: ${avg}%" >> $REPORT_FILE
    fi
done

# Memory trend
echo "### Average Memory Usage by Day" >> $REPORT_FILE
for i in {1..30}; do
    date=$(date -d "$i days ago" +%Y%m%d)
    if [ -f /var/log/sa/sa$(date -d "$i days ago" +%d) ]; then
        avg=$(sar -r -f /var/log/sa/sa$(date -d "$i days ago" +%d) | grep Average
| awk '{print $4}')
        echo "$date: ${avg}%" >> $REPORT_FILE
    fi
done

# Disk growth
echo "### Disk Usage Trend" >> $REPORT_FILE
df -h | grep -vE '^Filesystem|tmpfs' >> $REPORT_FILE

# Forecast
cat >> $REPORT_FILE << EOF

## Capacity Forecast (3 months)

Based on current trends:
- CPU: [Calculate growth rate]
- Memory: [Calculate growth rate]
- Disk: [Calculate growth rate]
```

```
## Recommendations
- [ ] Scale CPU if usage > 70%
- [ ] Add RAM if usage > 80%
- [ ] Expand disk if > 70% full
EOF

echo "Report generated: $REPORT_FILE"
```

6. Documentation and Runbooks

Performance Runbook Template

```
# Performance Investigation Runbook

## Quick Reference
| Issue Type | Primary Tool | Secondary Tool | Command |
|------------|--------------|----------------|---------|
| High CPU | htop | perf | `perf top -g` |
| Memory Leak | atop -m | bpftrace | `memleak-bpfcc -p PID` |
| Disk Slow | iostat -x | iotop | `iotop -o` |
| Network | sar -n | bpftrace | `tcpretrans-bpfcc` |

## Standard Operating Procedures

### SOP-001: High CPU Investigation
1. Identify process: `htop` (Sort by CPU with Shift+P)
2. Check CPU distribution: `sar -P ALL 1 10`
3. Profile hot functions: `perf record -p <PID> -g sleep 30`
4. Analyze: `perf report`
5. If kernel issue: `perf top -K`

### SOP-002: Memory Leak Detection
1. Monitor trend: `atop -m 5`
2. Check OOM: `dmesg | grep -i oom`
3. Identify leaking process: `ps aux --sort=-%mem | head`
4. Profile allocations: `bpftrace memleak.bt -p <PID>`
5. Generate heap dump (if Java): `jmap -dump:format=b,file=heap.bin <PID>`

### SOP-003: Disk Performance Issue
1. Check utilization: `iostat -x 1 10`
2. Identify processes: `iotop -o`
3. Check latency: `bpftrace biolatency.bt`
4. Filesystem check: `df -i` (inode usage)
5. RAID status: `cat /proc/mdstat`

## Escalation Criteria
- CPU >90% for >15 min → Escalate to Senior SRE
- Memory >95% → Immediate escalation
- Disk 100% util → Immediate escalation
- OOM kills → Immediate escalation
```

7. Monitoring as Code

Infrastructure as Code Example

```yaml
# monitoring-config.yml (Ansible playbook)
---
- name: Configure System Monitoring
  hosts: all
  become: yes
  tasks:
    - name: Install monitoring tools
      apt:
        name:
          - sysstat
          - atop
          - iotop
          - htop
          - glances
          - bpftrace
          - linux-tools-generic
        state: present
        update_cache: yes

    - name: Enable sysstat
      service:
        name: sysstat
        state: started
        enabled: yes

    - name: Configure sysstat intervals
      lineinfile:
        path: /etc/cron.d/sysstat
        regexp: '^\*/5'
        line: '*/5 * * * * root /usr/lib/sysstat/sa1 1 1'

    - name: Enable atop
      service:
        name: atop
        state: started
        enabled: yes

    - name: Configure atop retention
      lineinfile:
        path: /etc/default/atop
        regexp: '^LOGGENERATIONS'
        line: 'LOGGENERATIONS=30'

    - name: Deploy health check script
      copy:
        src: system-health-check.sh
        dest: /usr/local/bin/system-health-check.sh
        mode: '0755'
```

```
    - name: Schedule health checks
      cron:
        name: "System Health Check"
        minute: "*/5"
        job: "/usr/local/bin/system-health-check.sh"

    - name: Configure log rotation
      copy:
        dest: /etc/logrotate.d/monitoring
        content: |
          /var/log/health-check.log {
              daily
              rotate 30
              compress
              missingok
              notifempty
          }
```

8. Team Training and Knowledge Sharing

Recommended Learning Path

```
Week 1: Fundamentals
├── vmstat, iostat basics
├── htop navigation
└── Understanding system metrics

Week 2: Historical Analysis
├── sar deep dive
├── atop usage
└── Trend analysis

Week 3: Advanced Tracing
├── perf profiling
├── bpftrace scripts
└── Kernel tracing

Week 4: Automation
├── Monitoring scripts
├── Alert configuration
└── Incident response

Ongoing: Real Incidents
└── Learn from production issues
```

Özet ve Hızlı Başvuru

Günlük Rutin Kontroller

```
# Sabah kontrolü (5 dakika)
glances                          # Genel sağlık
htop                             # Process durumu
df -h                            # Disk kullanımı
systemctl status <services>      # Servis durumu

# Performans özeti
vmstat 1 10 | tail -5
iostat -x 1 5 | tail -10
sar -u -r -d 1 5

# Log kontrol
dmesg | tail -50
journalctl -p err -S today
```

Acil Durum Komutları

```
# Sistem donması / yüksek load
top -b -n 1 | head -20
ps aux --sort=-%cpu | head -10
killall -9 <process_name>      # Son çare!

# Bellek tükenme
ps aux --sort=-%mem | head -10
sync; echo 3 > /proc/sys/vm/drop_caches  # Cache temizle
pkill <memory_leaking_process>

# Disk dolu
du -sh /* | sort -h | tail -10
find /var/log -name "*.log" -mtime +7 -delete
journalctl --vacuum-size=100M

# Network sorunları
netstat -tunlp | grep ESTABLISHED | wc -l
ss -s
tcpdump -i any -c 100
```

Tool Comparison Cheat Sheet

Gereksinim	En İyi Tool	Alternatif
Quick overview	glances, htop	top
CPU profiling	perf	bpftrace
Memory analysis	atop	vmstat
Disk I/O	iostat, iotop	atop

Gereksinim	En İyi Tool	Alternatif
Historical data	sar, atop	logs
Network	sar -n	dstat
Real-time all-in-one	dstat	glances
Deep kernel tracing	bpftrace	perf
Process tree	htop -t	pstree
Container monitoring	glances --docker	docker stats