Complete Linux Server Monitoring & Troubleshooting Guide

A Comprehensive Technical Reference for System Administrators and DevOps Engineers

Mohamed Achraf Sabbagh

December 5 , 2025

Topics Covered:

CPU Monitoring • Memory Management
Disk I/O Analysis • Network Diagnostics
Performance Optimization • Troubleshooting Workflows

This guide provides production-ready monitoring strategies,
real-world troubleshooting scenarios, and best practices
for maintaining high-performance Linux server environments.

Contents

Abstract

This comprehensive guide provides system administrators, DevOps engineers, and infrastructure professionals with detailed methodologies for monitoring, diagnosing, and resolving performance issues in Linux server environments. Through practical examples, real-world scenarios, and systematic troubleshooting workflows, this document covers the four critical pillars of server health: CPU utilization, memory management, disk I/O performance, and network connectivity.

Each section includes command references, output interpretation, threshold guidelines, and actionable remediation strategies. The guide emphasizes holistic analysis, teaching readers to correlate metrics across subsystems to identify root causes rather than symptoms.

Keywords: Linux, System Administration, Performance Monitoring, Troubleshooting, DevOps, Infrastructure Management

1 Introduction

1.1 Purpose and Scope

In modern infrastructure environments, server performance directly impacts application availability, user experience, and business operations. This guide provides a systematic approach to:

- Monitor critical system resources in real-time
- Identify performance bottlenecks accurately
- Diagnose root causes of system degradation
- Implement effective remediation strategies
- Establish proactive monitoring frameworks

1.2 Target Audience

This guide is designed for:

- System Administrators managing Linux infrastructure
- DevOps Engineers responsible for application performance
- Site Reliability Engineers (SREs) maintaining production systems
- Infrastructure Engineers planning capacity
- Technical Support personnel diagnosing issues

1.3 How to Use This Guide

Key Information

Quick Reference: Each section includes:

- Command syntax and examples
- Output interpretation guidelines
- Health threshold matrices
- Real-world troubleshooting scenarios
- Remediation strategies

During an Incident: Jump directly to the relevant section's troubleshooting workflow.
For Learning: Read sequentially, practicing commands on test systems.
For Reference: Use the Quick Reference Cards at the end of each major section.

2 CPU Monitoring

2.1 Understanding CPU Architecture

Before implementing monitoring, understanding your system's CPU architecture is essential for accurate interpretation of metrics.

```
# Count total logical CPUs
nproc

# Detailed CPU information
lscpu

# Example output interpretation:
# CPU(s): 8                <- Total logical CPUs
# Thread(s) per core: 2    <- Hyperthreading enabled
# Core(s) per socket: 4    <- 4 physical cores
# Socket(s): 1             <- 1 physical CPU chip
# Result: 1 chip x 4 cores x 2 threads = 8 logical CPUs
```

Listing 1: CPU Architecture Discovery

Key Information

Why Architecture Matters:
A load average of 8.0 on an 8-core system indicates full utilization, while 16.0 indicates processes are queuing. Understanding this relationship is crucial for accurate capacity planning and performance assessment.

2.2 Load Average Analysis

Load average represents the average number of processes that are either running, runnable (waiting for CPU), or in uninterruptible sleep (waiting for I/O).

2.2.1 Command: uptime

```
$ uptime
 14:32:18 up 23 days, 4:12, 3 users, load average: 2.15, 1.98, 1.76
```

Listing 2: Checking Load Average

The three numbers represent 1-minute, 5-minute, and 15-minute averages respectively.

2.2.2 Load Average Interpretation Matrix

Table 1: Load Average Assessment Guidelines

Load vs Cores	Meaning	Action Required
Load $<$ Cores	Underutilized	Normal, capacity available
Load $\approx$ Cores	Fully utilized	Monitor trends
Load $= 1.5\times$ Cores	Moderate queuing	Investigate within minutes
Load $= 2\times$ Cores	Heavy queuing	Immediate investigation
Load $> 3\times$ Cores	Severe saturation	Emergency response

Example

Scenario A - 4-core system:

`load average: 1.20, 1.45, 1.68`

Analysis: All loads < 4.0 (healthy). Upward trend ($1.20 \rightarrow 1.68$) indicates recent activity increase. Continue monitoring.

Scenario B - 4-core system:

`load average: 12.50, 8.30, 4.20`

Analysis:

- 1-min: 12.50 (3× cores) - CRITICAL spike occurring now
- 5-min: 8.30 (2× cores) - Heavy load past 5 minutes
- 15-min: 4.20 (1× cores) - Was normal 15 minutes ago

Action: Immediate investigation with `top` or `htop`.

2.3 Per-Core CPU Utilization

2.3.1 Command: mpstat

```
$ mpstat -P ALL 1 5
# -P ALL: show all processors
# 1: 1 second interval
# 5: 5 iterations

Average: CPU    %usr  %nice  %sys %iowait %irq %soft %steal %idle
Average: all   15.23   0.00  4.12    2.34 0.00  0.50   0.00 77.81
Average:   0   45.67   0.00  6.21    0.45 0.00  0.12   0.00 47.55
Average:   1   12.34   0.00  3.88    1.22 0.00  0.45   0.00 82.11
Average:   2   10.11   0.00  3.45    3.56 0.00  0.67   0.00 82.21
Average:   3    9.87   0.00  3.02    4.11 0.00  0.56   0.00 82.44
```

Listing 3: Per-Core CPU Statistics

2.3.2 CPU Metrics Explained

%usr (User Space): Time executing user applications

- Normal: 20-70%
- High (>80%): CPU-intensive work (calculations, encoding)

%sys (System/Kernel): Time executing kernel operations

- Normal: 1-10%
- High (>20%): Excessive syscalls, context switching, or kernel work

%iowait: CPU idle waiting for I/O completion

- Normal: <5%
- Concerning: 10-20%

- Critical: >20% sustained

%steal (Virtual Machines Only): Time stolen by hypervisor

- Normal: <5%
- High (>10%): VM throttling or host oversold

Warning

Critical Indicator: If `%sys > %usr`, this often indicates a kernel bottleneck or inefficient I/O patterns. Investigate with `perf` or check connection counts and file operation patterns.

2.4 Process-Level CPU Analysis

2.4.1 Interactive Monitoring with top

```
top -o %CPU
# Press '1' to show individual cores
# Press 'H' to show threads
# Press 'c' to show full command line
# Press 'M' to sort by memory
```

Listing 4: Top Command Usage

2.4.2 Sample Output Interpretation

```
PID   USER      PR NI  VIRT   RES   SHR S %CPU %MEM  TIME+ COMMAND
3421  www-data  20  0  982M   89M   12M R 87.3  5.6 342:18 php-fpm:
   worker
5632  mysql     20  0 3.1G  1.8G   23M S 23.5 12.1 1892:34 mysqld
7821  root      20  0  156M   45M   8M  R 15.2  2.8  45:23 python3
   worker.py
```

Listing 5: Top Command Output

Key Columns:

- **%CPU:** Can exceed 100% (multi-threaded across cores)
- **TIME+:** Total CPU time consumed (342:18 = 342 minutes, 18 seconds)
- **S (State):** R=Running, S=Sleeping, D=Uninterruptible (I/O wait), Z=Zombie
- **RES:** Actual physical RAM used (most important memory metric)

2.5 CPU Saturation Detection

```
vmstat 1 5

procs -----------memory---------- --swap-- ---io-- -system- ---cpu---
 r  b   swpd   free   buff  cache   si   so    bi   bo   in   cs us sy id wa
 3  2 524288 445632 125K   3.4G     0    0    12  456  890 1245 25  5 68  2
12  0 524288 398234 125K   3.4G     0    0     4  123 1123 1834 62 12 24  2
```

Listing 6: Checking Run Queue Depth

Critical Columns:

r (runnable): Processes waiting for CPU time

- Healthy: r < number of cores
- Saturated: r > cores (Example: 12 runnable on 4-core = 8 queued)

b (blocked): Processes in uninterruptible sleep (I/O wait)

2.6 CPU Troubleshooting Workflow

1. **Quick Check:** `uptime` - Is load high?
2. **Identify Type:** `mpstat -P ALL 1 3` - CPU or I/O?
3. **Find Culprit:** `top -o %CPU` or `htop`
4. **Investigate Process:**

```
ps aux | grep <PID>          # Full command
lsof -p <PID>                # Open files/sockets
strace -c -p <PID>           # System call profile

```

5. **If I/O Wait High:** Proceed to Disk Monitoring section

3 Memory (RAM) Monitoring

3.1 Linux Memory Management Fundamentals

Key Information

Critical Concept: Linux aggressively uses free RAM for caching. A system showing "low free memory" is typically *healthy*, not problematic. The key metric is `available` memory, not `free`.

Memory Hierarchy:

1. Used by applications (cannot be reclaimed)
2. Cache/Buffers (kernel caching, can be freed instantly)
3. Free (truly unused)

3.2 Primary Memory Analysis

3.2.1 Command: free -h

```
$ free -h
               total        used        free      shared  buff/cache   available
Mem:            15Gi        11Gi       500Mi       120Mi       3.5Gi        3.6
   Gi
Swap:          2.0Gi       512Mi       1.5Gi
```

Listing 7: Memory Status Check

3.2.2 Column Interpretation

total: Physical RAM installed (15 GiB)

used: Memory used by applications + kernel (11 GiB)

free: Truly unused memory (500 MiB) - IGNORE for health

buff/cache: Kernel caching for performance (3.5 GiB) - Good!

available: MOST CRITICAL - Memory allocatable without swapping (3.6 GiB)

Swap used: Memory paged to disk (512 MiB)

3.3 Memory Health Assessment

Table 2: Memory Health Thresholds

Available %	Status	Action	Risk
$> 20\%$	Healthy	Normal operation	None
10-20%	Monitor	Watch trends	Low
5-10%	Warning	Identify memory hogs	Medium
$< 5\%$	Critical	Immediate action	High - OOM risk

Example

Scenario A - Healthy System:

```
Mem: 32Gi  28Gi  800Mi  200Mi  3.0Gi  3.5Gi (11% available)
```

Analysis: 28 GiB used, but 3.5 GiB available. Large cache (3.0 GiB) can be reclaimed. Status: Healthy.

Scenario C - Memory Pressure:

```
Mem:  8Gi  7.8Gi  50Mi  100Mi  150Mi  180Mi (2% available)
Swap: 4Gi  2.1Gi  1.9Gi
```

Analysis: Only 180 MiB available, actively swapping 2.1 GiB. Status: Critical - Add RAM or reduce usage immediately.

3.4 Identifying Memory-Hungry Processes

```
$ ps aux --sort=-rss | head -n 10

USER    PID %CPU %MEM    VSZ   RSS  COMMAND
mysql  1245 5.2 18.5 4523M 2987M /usr/sbin/mysqld
java   5632 3.1 15.2 8234M 2456M java -Xmx2G -jar app.jar
www    7821 1.2 12.3 2345M 1987M php-fpm: pool www
redis  3421 0.8  8.7 1234M 1405M redis-server *:6379
```

Listing 8: Top Memory Consumers

Key Metrics:

- **RSS (Resident Set Size):** Actual physical RAM used (most important)
- **VSZ (Virtual Size):** Usually irrelevant - includes shared libraries
- **%MEM:** Percentage of total RAM

3.5 Swap Analysis

3.5.1 Understanding Swap Usage

```
vmstat 1 5

procs --------memory------- --swap-- ---io---
 r  b  swpd   free   cache   si   so   bi   bo
 1  0 524288 445632 3456K    0    0   12  456  <- No swapping (good)
 1  0 524288 423156 3456K    0  234    8  234  <- Swapping out
 1  2 524288 398234 3456K  456    0    4  123  <- Swapping in (BAD)
 2  1 524288 375123 3456K  234  567   12  345  <- Both (CRITICAL)
```

Listing 9: Monitoring Swap Activity

Critical Columns:

si (swap in): KB/s read from swap to RAM - MOST CRITICAL

so (swap out): KB/s written from RAM to swap

Critical

Swap Thrashing: When `si` is high and sustained, the system spends more time swapping than working. This creates a "death spiral" where everything slows down exponentially. Users will report the system as "hanging" or "frozen."

3.6 Memory Troubleshooting Workflow

1. **Check Status:** `free -h` - Is available < 10%?
2. **Swap Activity:** `vmstat 1 5` - Is si/so > 0?
3. **Find Memory Hogs:** `ps aux -sort=-rss | head -n 20`
4. **Check for Leaks:**

```
watch -n 10 'ps aux | grep <process> | awk "{print \$6}"'
# If RSS continuously grows without plateau -> memory leak

```

5. **Review OOM History:** `dmesg | grep -i "killed process"`

4 Disk & Storage Monitoring

4.1 Understanding Storage Performance

Modern storage technologies have vastly different performance characteristics:

Table 3: Storage Performance Comparison

Type	IOPS	Sequential Throughput
HDD (7200 RPM)	80-160	100-200 MB/s
SATA SSD	10,000-90,000	500-600 MB/s
NVMe SSD	100,000-1,000,000+	3,000-7,000 MB/s

4.2 Disk Space Monitoring

4.2.1 Command: df -h

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        40G   28G   11G  73% /
/dev/sdb1       100G   92G  3.5G  96% /var
/dev/sdc1       500G  234G  241G  50% /home
tmpfs           7.8G  1.2G  6.6G  16% /run
```

Listing 10: Filesystem Space Usage

4.2.2 Filesystem Health Thresholds

Table 4: Disk Space Alert Levels

Use%	Status	Action	Risk
$<$ 70%	Healthy	Monitor trends	None
70-85%	Elevated	Plan cleanup/expansion	Low
85-95%	Warning	Act within days	Service failures
95-98%	Critical	Immediate action	High failure risk
$>$ 98%	Emergency	Urgent intervention	System instability

Warning

Why Filesystems Fail Before 100%:

- Reserved blocks (5% for root on ext4 default)
- Metadata overhead (inodes, journals, allocation tables)
- Application behavior (many apps fail ungracefully when writes fail)

4.3 Inode Usage Analysis

4.3.1 The Hidden Quota Problem

```
$ df -i
Filesystem      Inodes   IUsed    IFree  IUse% Mounted on
/dev/sda1      6553600  340000  6213600     6% /
/dev/sdb1      6553600  6553590       10   100% /var
/dev/sdc1     32768000  123456  32644544     1% /home
```

Listing 11: Checking Inode Usage

Critical

The 100% Inode Problem:
Filesystem: /dev/sdb1 has 55GB free space but 100% inodes used.
Result: Cannot create ANY new files despite available space!
Error Message:

```
$ touch test.txt
touch: cannot touch 'test.txt': No space left on device
```

Common Causes:

- Mail queues with millions of small files
- Many rotated log files (.gz archives)
- PHP/web session directories
- Node.js/Python cache directories

4.4 Disk I/O Performance

4.4.1 Command: iostat -x

```
$ iostat -x 1 5

Device: r/s   w/s   rkB/s  wkB/s  avgrq-sz  avgqu-sz  await  svctm  %util
sda      12.3  45.6   1234   8765     234.5      2.34   45.2    8.9   67.5
sdb     456.7  23.4  45678   2345     987.6     12.45  234.6    2.1   98.9
sdc       0.5   1.2     50    102     128.0      0.12    3.2    0.5    5.2
```

Listing 12: Detailed I/O Statistics

4.4.2 Critical I/O Metrics

r/s, w/s: Reads/writes per second (IOPS)

avgqu-sz: Average queue length - Key saturation metric

- $<$ 1: Disk keeping up
- 1-4: Moderate queue, acceptable
- $>$ 4: Requests backing up
- $>$ 10: Severe congestion

await: Average time (ms) for I/O requests

- HDD baseline: 5-15ms

- SSD baseline: <1ms
- > 20ms on SSD: Problem
- > 50ms on HDD: Severe problem
- > 100ms: Critical - users notice slowness

%util: Percentage of time with at least one I/O pending

- 100% on SSD ≠ saturated (handles parallel I/O)
- 100% on HDD = likely saturated

Example

Example 1: HDD Saturation

```
Device: r/s    w/s   avgqu-sz  await   %util
sda     234.5  89.3    8.45   156.7    98.5
```

Analysis:

- avgqu-sz=8.45: Many requests waiting
- await=156.7ms: Very slow (normal HDD 10ms)
- %util=98.5%: Saturated

Diagnosis: HDD cannot handle 323 IOPS (r/s + w/s)
Solutions: Migrate to SSD, optimize queries, RAID configuration, move I/O-heavy workloads

4.5 SMART Health Monitoring

```
# Quick health check
smartctl -H /dev/sda

# Full SMART data
smartctl -a /dev/sda
```

Listing 13: Checking Disk Health

4.5.1 Critical SMART Attributes

Table 5: Critical SMART Indicators

Attribute	Threshold	Action
Reallocated_Sector_Ct	> 0	Plan replacement
	> 10	Imminent failure
Current_Pending_Sector	> 0	Monitor closely
	> 5	Replace soon
Offline_Uncorrectable	> 0	Data loss risk
Reported_Uncorrect	> 0	Reliability concern
Power_On_Hours	> 40,000	Consider age in planning

Critical

Immediate Replacement Indicators:

- Reallocated_Sector_Ct > 10
- Current_Pending_Sector > 5
- Offline_Uncorrectable > 0
- Any combination of growing errors

Action: Immediate backup, snapshot data, and schedule replacement.

4.6 Disk Troubleshooting Workflow

1. **Check Space:** `df -h` and `df -i`
2. **If Space Issue:**

```
du -sh /* | sort -rh
find / -type f -size +1G -exec ls -lh {} \;

```

3. **Check I/O:** `iostat -x 1 5`
4. **If High I/O:** `iotop -o -a`
5. **Check Health:** `smartctl -a /dev/sda`

5 Network Monitoring

5.1 Network Monitoring Layers

Network troubleshooting requires a layered approach:

1. **Physical:** Cables, NICs, link speed
2. **Data Link:** Ethernet frames, errors, drops
3. **Network:** IP packets, routing, latency
4. **Transport:** TCP connections, UDP packets
5. **Application:** HTTP, SSH, Database protocols

5.2 Interface Traffic Analysis

5.2.1 Command: ip -s link

```
$ ip -s link show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    link/ether 00:1a:2b:3c:4d:5e brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    98765432109 87654321 0      523     0       12345
    TX: bytes  packets  errors  dropped carrier collsns
    45678901234 56789012 0      127     0       0
```

Listing 14: Interface Statistics

5.2.2 Counter Interpretation

errors: Malformed frames (CRC, alignment, framing)

- $>$ 0.01% of packets: Investigate physical layer
- Causes: Bad cable, faulty NIC, switch port issue

dropped: Packets discarded by kernel

- RX dropped: Receive buffer overflow
- TX dropped: Transmit queue full
- $>$ 0: Tune ring buffers or investigate driver

overrun: NIC buffer overflow

- $>$ 0: Severe hardware bottleneck (rare)

carrier: Carrier signal lost (Layer 1 issue)

- $>$ 0: Check physical connection, cable, transceiver

5.3 Connection Tracking

5.3.1 Command: ss (Socket Statistics)

```
$ ss -s
Total: 1247 (kernel 1532)
TCP:   456 (estab 387, closed 45, orphaned 2, timewait 38)
UDP:   89

# List listening ports
$ ss -tlnp
State  Recv-Q Send-Q Local Address:Port  Process
LISTEN 0      128    0.0.0.0:22          users:(("sshd",pid=1234))
LISTEN 0      128    0.0.0.0:80          users:(("nginx",pid=5678))
LISTEN 0      80     127.0.0.1:3306      users:(("mysqld",pid=9012))
```

Listing 15: Connection Summary

5.3.2 TCP Connection States

LISTEN: Server socket waiting for connections

- 0.0.0.0 = listening on all interfaces
- 127.0.0.1 = localhost only (secure)

ESTAB (Established): Active connection

- High count may indicate heavy traffic or connection leak

TIME-WAIT: Connection closed, waiting for final ACK

- Normal: lasts 60 seconds (2× MSL)
- Many TIME-WAIT (thousands): High connection churn
- Solution: Tune `net.ipv4.tcp_tw_reuse=1`

CLOSE-WAIT: Remote closed, local app hasn't closed

- Growing CLOSE-WAIT: Application bug (not closing sockets)
- Solution: Fix application code, restart service

5.4 Latency and Packet Loss Analysis

5.4.1 Basic Connectivity: ping

```
$ ping -c 10 -i 0.2 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=12.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=11.8 ms
...
--- 8.8.8.8 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss
rtt min/avg/max/mdev = 11.756/12.437/13.145/0.421 ms
```

Listing 16: Latency Testing

Latency Guidelines:

- Local network: <1ms
- Same city: 1-10ms
- Same country: 10-50ms
- Intercontinental: 100-300ms

Packet Loss Impact:

- 0%: Perfect
- <1%: Acceptable
- 1-5%: Noticeable degradation (VoIP suffers)
- >5%: Severe problems

5.4.2 Advanced Path Analysis: mtr

```
$ mtr -r -c 100 -n example.com

HOST                Loss%   Snt   Last   Avg   Best  Wrst  StDev
1. 10.0.0.1          0.0%   100    0.4   0.5    0.3   1.2   0.1
2. 203.0.113.1       0.0%   100   10.2  10.5    9.8  15.3   0.8
3. 198.51.100.45     2.0%   100   20.1  22.3   19.5  80.4   8.7
4. 192.0.2.12        2.0%   100   21.5  23.1   20.2  82.1   9.1
5. 93.184.216.34     0.0%   100   22.3  23.5   21.8  28.9   1.2
```

Listing 17: Multi-hop Trace Route

Key Information

Interpreting mtr Results:
Loss at single hop, but not later: Often false positive (router rate-limiting ICMP)
Loss at hop AND all subsequent hops: Real problem at that hop
In example above: 2% loss starts at hop 3 and persists → Root cause at hop 3

5.5 Network Troubleshooting Workflow

1. **Physical Layer:**

```
ethtool eth0                 # Link up? Speed correct?
ip link show eth0            # Interface UP?

```

2. **Data Link Layer:**

```
ip -s link show eth0         # Errors? Drops?
ethtool -S eth0              # Detailed NIC stats

```

3. **Network Layer:**

```
ip route                     # Default route correct?
ping 8.8.8.8                 # Internet reachable?
ping <gateway>               # Local gateway reachable?

```

4. **Transport Layer:**

```
ss -s                      # Connection states normal?
ss -tlnp                   # Services listening?

```

5. **Application Layer:**

```
curl -I http://localhost  # App responding?
journalctl -u <service>   # App logs

```

6 Quick Reference Cards

6.1 CPU Quick Reference

Commands:

```
uptime                  # Load averages
top                     # Interactive monitor
htop                    # Better alternative
mpstat -P ALL 1         # Per-core stats
ps aux --sort=-%cpu     # Top CPU processes
```

Key Metrics:

- Load $<$ cores $=$ OK
- %iowait $>$ 10% $=$ check disk
- %steal $>$ 10% $=$ VM throttled

6.2 Memory Quick Reference

Commands:

```
free -h                      # Memory overview
ps aux --sort=-rss           # Top memory users
vmstat 1 5                   # Swap activity
smem -tk                     # Detailed breakdown
```

Key Metrics:

- available $<$ 10% $=$ warning
- swap si/so $>$ 0 $=$ problem
- OOM kills $=$ critical

6.3 Disk Quick Reference

Commands:

```
df -h                   # Disk space
df -i                   # Inode usage
du -sh /*               # Directory sizes
iostat -x 1 5           # I/O performance
iotop -o                # Top I/O processes
smartctl -a /dev/sda    # SMART health
```

Key Metrics:

- Space > 90% = act soon
- await > 20ms SSD = problem
- await > 50ms HDD = problem
- %util 100% HDD = saturated

6.4 Network Quick Reference

Commands:

```
ss -s                   # Connection summary
ss -tlnp                # Listening services
ip -s link              # Interface stats
iftop -i eth0           # Bandwidth per connection
mtr <destination>       # Path analysis
ping -c 10 <host>       # Basic connectivity
```

Key Metrics:

- Packet loss > 1% = issue
- Latency variation = jitter
- errors/drops > 0 = investigate
- TIME-WAIT high = tune reuse

7 Conclusion

Effective Linux server monitoring and troubleshooting requires:

- **Systematic methodology** - Follow structured workflows rather than random investigation
- **Holistic analysis** - Correlate metrics across CPU, memory, disk, and network
- **Understanding context** - Know your baseline and recognize deviations
- **Proactive monitoring** - Detect issues before they impact users
- **Continuous learning** - Document incidents and build organizational knowledge

Remember: The goal is not just to fix problems,
but to understand systems deeply enough
to prevent problems before they occur.

Appendix A: Essential Commands Cheat Sheet

Table 6: *
CPU Monitoring Commands

Command	Purpose
`uptime`	Display load averages
`top`	Interactive process viewer
`htop`	Enhanced interactive viewer
`mpstat -P ALL 1`	Per-core CPU statistics
`ps aux -sort=-%cpu`	List processes by CPU usage
`vmstat 1 5`	System statistics including run queue
`pidstat 1`	Per-process statistics over time
`perf top`	Real-time profiling

Table 7: *
Memory Monitoring Commands

Command	Purpose
`free -h`	Memory usage summary
`vmstat 1 5`	Memory and swap statistics
`ps aux -sort=-rss`	List processes by memory usage
`pmap -x <PID>`	Memory map of specific process
`smem -tk`	Detailed memory breakdown
`slabtop`	Kernel slab cache information
`cat /proc/meminfo`	Detailed memory statistics

Table 8: *
Disk Monitoring Commands

Command	Purpose
`df -h`	Filesystem space usage
`df -i`	Inode usage
`du -sh /*`	Directory sizes
`iostat -x 1 5`	Disk I/O statistics
`iotop -o`	Per-process I/O usage
`smartctl -a /dev/sda`	SMART disk health data
`lsblk`	List block devices
`blkid`	Block device attributes

Table 9: *
Network Monitoring Commands

Command	Purpose
`ss -s`	Socket statistics summary
`ss -tlnp`	Listening TCP sockets
`ip -s link`	Interface statistics
`ip route`	Routing table
`ethtool eth0`	NIC settings and status
`iftop -i eth0`	Real-time bandwidth usage
`nethogs`	Per-process bandwidth
`mtr <host>`	Network path analysis
`tcpdump -i eth0`	Packet capture
`nmap`	Network scanning