Independent Skill Development Mission

Independent Skill Development Mission NXT
EMPOWERING FUTURES, SHAPING TOMORROW

ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

PERFORMANCE TUNING & OPTIMIZATION

CHAPTER 1: INTRODUCTION TO PERFORMANCE TUNING AND OPTIMIZATION

What is Performance Tuning?

Performance tuning is the **process of optimizing system resources**, ensuring efficient operation, and improving the **speed, reliability, and scalability** of applications and servers. It involves analyzing **CPU, memory, disk I/O, and network utilization** to identify bottlenecks and implement corrective measures.

In cloud computing and DevOps environments, performance tuning is essential to **enhance system responsiveness, reduce latency, and lower operational costs**. Poorly optimized systems lead to **slow application performance, increased downtime, and higher infrastructure expenses**.

Example:
A **web application** experiencing slow response times due to **high CPU usage** on the backend server can be optimized by **load balancing, caching strategies, and reducing unnecessary background processes**.

CHAPTER 2: SYSTEM RESOURCE MONITORING AND ANALYSIS

ISDM-NXT

1. Monitoring System Performance Using Linux Tools

To optimize system performance**, monitoring tools** help identify **resource utilization and bottlenecks**.

Tool	Purpose
top/htop	Monitors CPU and memory usage in real-time.
vmstat	Provides detailed reports on memory, I/O, and CPU load.
iostat	Analyzes disk performance and I/O operations.
sar	Collects system performance data over time.
netstat/ss	Monitors network connections and packet transmission.

Example Usage:
To check CPU and memory usage using htop:

htop

To monitor disk performance:

iostat -x 5

These commands help **detect performance bottlenecks** in real-time.

2. Identifying Performance Bottlenecks

Performance bottlenecks occur when **a specific resource limits the overall system performance**.

Bottleneck	Causes	Solutions

ISDM-NXT

CPU Overload	High process execution, inefficient code	Optimize queries, use load balancing
Memory Exhaustion	Excessive caching, memory leaks	Increase RAM, enable swap memory
Disk I/O Delay	Heavy read/write operations	Use SSDs, optimize file system
Network Latency	High traffic, misconfigured firewall	Implement CDN, enable compression

---

CHAPTER 3: CPU PERFORMANCE TUNING

1. Managing CPU Utilization

High CPU usage affects application responsiveness. The top or htop command helps identify processes consuming excess CPU.

Example:
To view processes sorted by CPU usage:

top -o %CPU

Optimization Techniques:

- **Use process scheduling (nice & renice):**
    - Assign **lower priority** to less critical tasks:
    - renice +10 -p <PID>
- **Enable CPU throttling for non-critical services:**
- cpulimit -p <PID> -l 50
- **Optimize background processes using cgroups:**

ISDM-NXT

- cgcreate -g cpu:/limitedgroup
- cgclassify -g cpu:/limitedgroup <PID>

CHAPTER 4: MEMORY OPTIMIZATION TECHNIQUES

1. Analyzing Memory Usage

To check available and used memory:

free -m

To list memory-intensive processes:

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

2. Optimizing RAM and Swap Usage

- **Enable swap memory for improved performance:**
- sudo fallocate -l 2G /swapfile
- sudo chmod 600 /swapfile
- sudo mkswap /swapfile
- sudo swapon /swapfile
- **Clear cached memory without affecting running processes:**
- sync; echo 3 | sudo tee /proc/sys/vm/drop_caches
- **Use ZRAM for better memory compression:**
- sudo apt install zram-tools -y
- sudo systemctl enable zram-swap

ISDM-NXT

CHAPTER 5: DISK I/O PERFORMANCE OPTIMIZATION

1. Monitoring Disk I/O

Disk performance can be measured using:

iostat -x 5

To check the disk read/write speed:

dd if=/dev/zero of=/tmp/testfile bs=1M count=1000 oflag=direct

2. Optimizing Disk Usage

- **Use SSDs instead of HDDs for faster read/write operations.**
- **Optimize file system performance using tuning tools:**
- sudo tune2fs -o journal_data_writeback /dev/sda1
- **Enable write-back caching for faster disk writes:**
- sudo hdparm -W1 /dev/sda

CHAPTER 6: NETWORK OPTIMIZATION

1. Monitoring Network Performance

Use the netstat or ss command to analyze network traffic:

ss -tulnp

To measure latency:

ping -c 5 google.com

2. Improving Network Performance

- **Enable TCP Fast Open for better latency:**

ISDM-NXT

- echo 3 | sudo tee /proc/sys/net/ipv4/tcp_fastopen
- **Optimize network buffer sizes:**
- sudo sysctl -w net.core.rmem_max=16777216
- sudo sysctl -w net.core.wmem_max=16777216

CHAPTER 7: APPLICATION-LEVEL OPTIMIZATION

1. Database Performance Optimization

- **Index frequently used database queries.**
- **Enable query caching in MySQL:**
- SET GLOBAL query_cache_size = 1000000;
- **Analyze slow queries using MySQL slow query log:**
- sudo nano /etc/mysql/my.cnf

Add:

slow_query_log = 1

slow_query_log_file = /var/log/mysql-slow.log

long_query_time = 2

2. Web Server Performance Tuning

- **Enable Gzip compression in Nginx:**
- gzip on;
- gzip_types text/plain application/json;
- **Use a Content Delivery Network (CDN) for static files.**

ISDM-NXT

- **Optimize caching with Redis or Memcached.**

CHAPTER 8: CASE STUDY – PERFORMANCE TUNING FOR AN E-COMMERCE WEBSITE

Scenario:

An **e-commerce website** experiences **slow page load times**, high **server CPU usage**, and **frequent downtime** during peak hours.

Solution Using Performance Tuning Techniques:

1. **Monitor CPU and memory usage** using top, htop, and vmstat.
2. **Optimize the database** by indexing queries and enabling caching.
3. **Reduce disk I/O delays** using SSDs and filesystem tuning.
4. **Improve network performance** with TCP optimizations.
5. **Use a load balancer** to distribute traffic across multiple servers.

Outcome:

- **Page load times reduced by 50%.**
- **CPU and memory usage optimized, preventing server crashes.**
- **Increased uptime and reliability during high traffic periods.**

CHAPTER 9: EXERCISE

1. **Use htop and iostat to analyze CPU and disk usage.**

ISDM-NXT

2. **Optimize memory by enabling swap space.**
3. **Tune database queries for better performance.**
4. **Implement caching in a web server using Nginx.**
5. **Measure network latency and apply TCP optimizations.**

CONCLUSION

Performance tuning is **essential for optimizing Linux servers, applications, and cloud environments**. By applying **CPU, memory, disk, network, and application-level optimizations**, DevOps engineers can **ensure high availability, low latency, and cost-effective system operations**.

ISDM-NXT

KERNEL TUNING FOR BETTER PERFORMANCE

CHAPTER 1: INTRODUCTION TO KERNEL TUNING

What is Kernel Tuning?

Kernel tuning is the **process of optimizing the Linux kernel parameters** to improve system performance, scalability, and resource efficiency. The Linux kernel is the **core of the operating system**, managing hardware, processes, memory, and I/O operations. By adjusting kernel parameters, we can **enhance CPU scheduling, memory management, network performance, and disk I/O efficiency**.

Kernel tuning is particularly useful in **high-performance computing (HPC), cloud computing, database servers, and real-time systems**, where default kernel settings may not be sufficient to handle intensive workloads.

Example:
A **database server** experiencing high latency and frequent slowdowns can benefit from kernel tuning by **optimizing memory allocation, increasing buffer sizes, and adjusting CPU scheduling policies**.

ISDM-NXT

CHAPTER 2: UNDERSTANDING KERNEL PARAMETERS AND SYSCTL

1. What is sysctl?

sysctl is a command-line utility that allows administrators to **modify kernel parameters** at runtime without rebooting. These parameters are stored in /proc/sys/ and can be viewed or modified using sysctl.

2. Viewing Kernel Parameters

To list all current kernel parameters:

sysctl -a

To view a specific parameter:

sysctl net.ipv4.tcp_rmem

3. Modifying Kernel Parameters Temporarily

To change a parameter for the current session:

sysctl -w net.core.somaxconn=1024

This change **does not persist** after reboot.

4. Making Kernel Changes Permanent

To apply changes permanently, add them to /etc/sysctl.conf:

echo "net.core.somaxconn=1024" | sudo tee -a /etc/sysctl.conf

sudo sysctl -p

CHAPTER 3: CPU SCHEDULING AND PROCESS MANAGEMENT TUNING

1. Understanding CPU Scheduling

The Linux kernel manages CPU allocation using different **scheduler policies**, including:

- **CFS (Completely Fair Scheduler)** – Default for balancing CPU load.
- **RT (Real-Time Scheduler)** – Used for low-latency applications.

2. Adjusting Process Priorities

ISDM-NXT

To modify CPU priority dynamically:

- Lower the priority (increase nice value):
- renice +10 -p <PID>
- Increase priority (requires root privileges):
- renice -10 -p <PID>

3. Changing Default Scheduler Policy

To use a **real-time scheduler,** edit GRUB configuration:

sudo nano /etc/default/grub

Modify:

GRUB_CMDLINE_LINUX_DEFAULT="quiet splash sched_rt_runtime=950000"

Then update GRUB:

sudo update-grub && sudo reboot

CHAPTER 4: MEMORY MANAGEMENT OPTIMIZATION

1. Understanding Linux Memory Management

The Linux kernel uses different techniques to **manage RAM efficiently,** including:

- **Page caching** – Stores frequently accessed data in memory.
- **Swapping** – Moves inactive memory pages to disk.
- **OOM (Out-of-Memory) Killer** – Kills processes when RAM is exhausted.

ISDM-NXT

2. Optimizing Swap Usage

To check current swap settings:

sysctl vm.swappiness

To reduce excessive swapping (recommended for database servers):

sysctl -w vm.swappiness=10

Make it permanent:

echo "vm.swappiness=10" | sudo tee -a /etc/sysctl.conf

sudo sysctl -p

3. Adjusting Dirty Page Writeback

To optimize disk writes, modify dirty page settings:

sysctl -w vm.dirty_ratio=20

sysctl -w vm.dirty_background_ratio=10

These settings **reduce the frequency of disk writes,** improving performance.

CHAPTER 5: DISK I/O OPTIMIZATION

1. Monitoring Disk Performance

To check disk usage statistics:

iostat -x 5

2. Changing I/O Schedulers

Linux provides different I/O schedulers:

ISDM-NXT

- **CFQ (Completely Fair Queuing)** – Default for desktops.
- **Deadline** – Optimized for database servers.
- **noop** – Best for SSDs and virtualized environments.

To check the current scheduler:

cat /sys/block/sda/queue/scheduler

To change the scheduler to deadline:

echo deadline | sudo tee /sys/block/sda/queue/scheduler

To make it persistent:

echo 'GRUB_CMDLINE_LINUX="elevator=deadline"' | sudo tee -a /etc/default/grub

sudo update-grub && sudo reboot

CHAPTER 6: NETWORK PERFORMANCE TUNING

1. Increasing Maximum Network Connections

By default, Linux limits the number of open sockets. To increase it:

sysctl -w net.core.somaxconn=2048

2. Optimizing TCP Buffers

To improve TCP performance for high-speed networks:

sysctl -w net.ipv4.tcp_rmem="4096 87380 6291456"

sysctl -w net.ipv4.tcp_wmem="4096 87380 6291456"

3. Enabling TCP Fast Open

ISDM-NXT

To reduce handshake time in TCP connections:

sysctl -w net.ipv4.tcp_fastopen=3

Chapter 7: Kernel Security Hardening

1. Disabling IP Forwarding

If not needed, disable IP forwarding for security:

sysctl -w net.ipv4.ip_forward=0

2. Enabling SYN Cookies to Prevent DoS Attacks

sysctl -w net.ipv4.tcp_syncookies=1

3. Restricting Core Dumps

sysctl -w fs.suid_dumpable=0

Chapter 8: Case Study – Performance Tuning for a High-Traffic Web Server

Scenario:

A **news website** running on an **Nginx web server** experiences **slow response times and high CPU load during peak traffic**.

Solution Using Kernel Tuning:

1. **Optimize CPU scheduling** by increasing process priority for Nginx.
2. **Tune memory management** to reduce swapping.

ISDM-NXT

3. **Increase network connection limits** to handle high traffic loads.
4. **Use the "deadline" I/O scheduler** to improve disk read/write efficiency.

Outcome:

- **50% improvement in page load times**.
- **Lower CPU and memory usage** during peak traffic hours.
- **Increased server uptime and better user experience**.

CHAPTER 9: EXERCISE

1. **Use sysctl to check current kernel parameters.**
2. **Modify the CPU scheduler for performance improvement.**
3. **Adjust memory management settings to reduce swap usage.**
4. **Optimize TCP buffer sizes for better network performance.**
5. **Change the I/O scheduler and analyze its impact on disk performance.**

CONCLUSION

Kernel tuning is a **powerful technique** for optimizing **CPU performance, memory management, disk I/O, and network efficiency**. By applying **sysctl optimizations**, system administrators and DevOps engineers can **enhance system performance, reduce latency, and ensure high availability for critical applications**.

ISDM-NXT

UNDERSTANDING LOAD BALANCING IN LINUX

CHAPTER 1: INTRODUCTION TO LOAD BALANCING

What is Load Balancing?

Load balancing is the **process of distributing incoming network traffic** across multiple servers to ensure high availability, reliability, and optimal performance. In Linux environments, load balancing is used to prevent **server overload**, improve **scalability**, and ensure **fault tolerance** in web applications, databases, and network services.

Why is Load Balancing Important?

- **Enhances availability:** Prevents system failures by distributing load.
- **Improves performance:** Ensures optimal response times for users.
- **Provides redundancy:** Ensures failover in case of server failure.
- **Scales applications:** Allows seamless scaling of cloud-based services.

Example Scenario

A **high-traffic e-commerce website** distributes its traffic across **multiple web servers** using a Linux-based **load balancer (HAProxy, Nginx, or IPVS)** to ensure fast response times during peak sales events.

CHAPTER 2: TYPES OF LOAD BALANCING IN LINUX

1. Network Load Balancing

ISDM-NXT

Network-based load balancing distributes traffic at the **network layer (Layer 3 & Layer 4)** to **optimize packet routing and prevent network congestion**.

Example: Load Balancing Using IPVS (IP Virtual Server)

```
ipvsadm -A -t 192.168.1.100:80 -s rr
ipvsadm -a -t 192.168.1.100:80 -r 192.168.1.101:80 -m
ipvsadm -a -t 192.168.1.100:80 -r 192.168.1.102:80 -m
```

Here, traffic for **192.168.1.100** (virtual IP) is distributed to **192.168.1.101 and 192.168.1.102**.

2. Application Load Balancing

Application-layer (Layer 7) load balancers distribute traffic based on **URL path, HTTP headers, cookies, or session data**.

Example: Load Balancing Using Nginx

```
upstream backend_servers {
   server 192.168.1.101;
   server 192.168.1.102;
}

server {
   listen 80;
   location / {
      proxy_pass http://backend_servers;
   }
}
```

ISDM-NXT

This configuration **routes traffic to two backend servers**.

3. Hardware vs. Software Load Balancing

Type	Description	Example
Hardware Load Balancer	Dedicated physical device for distributing traffic.	F5, Citrix ADC
Software Load Balancer	Open-source or proprietary software running on Linux.	HAProxy, Nginx, IPVS

CHAPTER 3: LOAD BALANCING ALGORITHMS

1. Round Robin

- Distributes requests **sequentially** across all servers.
- Best for **equal-capacity servers**.

```
ipvsadm -A -t 192.168.1.100:80 -s rr
```

2. Least Connections

- Directs traffic to the **server with the fewest active connections**.
- Best for **uneven workloads**.

```
upstream backend {
  least_conn;
  server 192.168.1.101;
  server 192.168.1.102;
}
```

3. IP Hash

ISDM-NXT

- Routes requests based on **client IP** for **session persistence**.

```
upstream backend {
  ip_hash;
  server 192.168.1.101;
  server 192.168.1.102;
}
```

4. Weighted Load Balancing

- Assigns a **higher weight to powerful servers**.

```
upstream backend {
  server 192.168.1.101 weight=3;
  server 192.168.1.102 weight=1;
}
```

CHAPTER 4: SETTING UP LOAD BALANCING WITH NGINX

1. Install Nginx on the Load Balancer Node

```
sudo apt update
sudo apt install nginx -y
```

2. Configure Nginx for Load Balancing

Edit the configuration file:

```
sudo nano /etc/nginx/nginx.conf
```

Add the following configuration:

```
http {
```

ISDM-NXT

```
    upstream web_servers {
        server 192.168.1.101;
        server 192.168.1.102;
    }

    server {
        listen 80;
        location / {
            proxy_pass http://web_servers;
        }
    }
}
```

3. Restart Nginx to Apply Changes

sudo systemctl restart nginx

4. Verify Load Balancing

Run the following command multiple times:

curl http://<load_balancer_IP>

The response should cycle between **backend servers**.

CHAPTER 5: LOAD BALANCING USING HAPROXY

1. Install HAProxy

sudo apt install haproxy -y

ISDM-NXT

2. Configure HAProxy for Load Balancing

Edit the HAProxy configuration file:

sudo nano /etc/haproxy/haproxy.cfg

Add the following:

```
frontend http_front
   bind *:80
   default_backend web_servers

backend web_servers
   balance roundrobin
   server server1 192.168.1.101:80 check
   server server2 192.168.1.102:80 check
```

3. Restart HAProxy

sudo systemctl restart haproxy

4. Verify HAProxy Load Balancing

Check HAProxy statistics:

curl http://<haproxy_IP>

CHAPTER 6: LOAD BALANCER HEALTH CHECKS AND FAILOVER

1. Configuring Health Checks in HAProxy

HAProxy continuously monitors backend servers using health checks. Modify the backend configuration:

```
backend web_servers
```

ISDM-NXT

```
balance roundrobin
server server1 192.168.1.101:80 check fall 3 rise 2
server server2 192.168.1.102:80 check fall 3 rise 2
```

- **fall 3** → Marks server **down after 3 failures**.
- **rise 2** → Brings server **back online after 2 successful checks**.

2. Enabling Health Checks in Nginx

```
upstream backend {
  server 192.168.1.101 max_fails=3 fail_timeout=30s;
  server 192.168.1.102 max_fails=3 fail_timeout=30s;
}
```

This prevents **unhealthy servers** from receiving traffic.

CHAPTER 7: CASE STUDY – LOAD BALANCING A WEB APPLICATION

Scenario:

A **video streaming service** faces **downtime issues** due to high traffic. They need a **scalable and reliable** solution to **balance traffic across multiple backend servers**.

Solution Using Linux Load Balancing:

1. **Deploy an Nginx load balancer** to distribute traffic across backend servers.
2. **Implement weighted round-robin** to prioritize high-performance servers.
3. **Enable health checks** to prevent failed servers from receiving traffic.

ISDM-NXT

4. **Use HAProxy for session persistence** for streaming continuity.

Outcome:

- **Reduced server downtime by 80%.**
- **Increased application performance during peak hours.**
- **Seamless user experience with high availability.**

CHAPTER 8: EXERCISE

1. **Set up Nginx as a load balancer and configure two backend servers.**
2. **Implement round-robin load balancing using HAProxy.**
3. **Configure health checks for backend servers.**
4. **Use weighted load balancing for efficient traffic distribution.**
5. **Monitor HAProxy statistics to analyze load distribution.**

CONCLUSION

Load balancing in Linux is **crucial for scaling applications**, ensuring **high availability**, and **improving system reliability**. By using **Nginx, HAProxy, or IPVS**, administrators can **distribute traffic efficiently**, enhance **server redundancy**, and maintain **optimal performance** in high-traffic environments.

ISDM-NXT

PROFILING LINUX APPLICATIONS

CHAPTER 1: INTRODUCTION TO PROFILING LINUX APPLICATIONS

What is Application Profiling?

Application profiling is the **process of analyzing the performance, resource utilization, and behavior** of an application running on a Linux system. Profiling helps **identify bottlenecks, optimize resource usage, and improve overall application efficiency**. It is particularly useful for **debugging slow applications, optimizing CPU and memory usage, and improving response times in production environments**.

Why is Profiling Important?

- **Identifies performance bottlenecks** in applications.
- **Improves CPU and memory efficiency** by detecting high resource-consuming processes.
- **Optimizes disk I/O and network performance** for scalable applications.
- **Enhances debugging** by tracking system calls and code execution.

Example Scenario

A **database-driven web application** experiences **high CPU usage and slow query execution**. Profiling tools such as **perf and strace** help detect inefficient database queries and unnecessary CPU cycles, leading to performance improvements.

CHAPTER 2: TYPES OF PROFILING IN LINUX

1. CPU Profiling

ISDM-NXT

Measures **CPU utilization** by analyzing which functions consume the most processing power.
Tools: perf, gprof, top, htop

2. Memory Profiling

Identifies **memory leaks** and inefficient memory usage.
Tools: valgrind, massif, smem, pmap

3. Disk I/O Profiling

Monitors **disk read/write performance** to identify slow storage operations.
Tools: iostat, iotop, blktrace

4. Network Profiling

Analyzes **network performance** to detect slow connections or high latency.
Tools: iftop, netstat, tcpdump, ss

5. Application Execution Profiling

Examines **application behavior, execution flow, and function calls**.
Tools: strace, ltrace, gdb

CHAPTER 3: CPU PROFILING USING PERF

1. Installing perf

sudo apt install linux-tools-common linux-tools-generic -y # Ubuntu

sudo yum install perf -y # CentOS/RHEL

2. Running perf to Profile CPU Usage

perf stat -p <PID>

ISDM-NXT

This command shows **CPU cycles, cache misses, and execution time** for a running process.

3. Generating a Performance Report

To capture performance data for **10 seconds**:

perf record -p <PID> -g -- sleep 10

perf report

This helps **identify functions consuming the most CPU cycles**.

CHAPTER 4: MEMORY PROFILING USING VALGRIND

1. Installing Valgrind

sudo apt install valgrind -y # Ubuntu/Debian

sudo yum install valgrind -y # CentOS/RHEL

2. Detecting Memory Leaks

Run an application with Valgrind:

valgrind --leak-check=full ./my_application

This command identifies **memory leaks** and reports **unfreed memory allocations**.

3. Profiling Heap Memory Usage

Use massif to track **heap allocation over time**:

valgrind --tool=massif ./my_application

To visualize the memory profile:

ms_print massif.out.*

ISDM-NXT

CHAPTER 5: DISK I/O PROFILING USING IOTOP AND IOSTAT

1. Installing iotop and iostat

sudo apt install iotop sysstat -y

2. Monitoring Disk Usage with iotop

sudo iotop

This command displays **real-time disk activity** and identifies processes with **high I/O usage**.

3. Checking Disk Performance with iostat

iostat -x 5

- r/s ⟶ Read operations per second
- w/s ⟶ Write operations per second
- await ⟶ Average wait time for I/O requests

4. Tracing Disk Activity with blktrace

sudo blktrace -d /dev/sda

This tool **monitors low-level disk operations**.

CHAPTER 6: NETWORK PROFILING USING IFTOP AND TCPDUMP

1. Installing Network Profiling Tools

sudo apt install iftop tcpdump -y

2. Monitoring Network Bandwidth with iftop

sudo iftop -i etho

This command shows **real-time network traffic**.

3. Capturing Network Packets with tcpdump

ISDM-NXT

To capture all packets on etho:

sudo tcpdump -i etho

To capture packets from a specific IP:

sudo tcpdump -i etho src 192.168.1.10

CHAPTER 7: APPLICATION EXECUTION PROFILING USING STRACE AND LTRACE

1. Tracing System Calls with strace

To trace system calls made by a process:

strace -p <PID>

To measure **execution time of each system call**:

strace -c ./my_application

2. Tracing Library Calls with ltrace

To trace library function calls:

ltrace ./my_application

CHAPTER 8: CASE STUDY – PROFILING A HIGH-CPU WEB APPLICATION

Scenario:

A **Python-based web application** running on a **Linux server** experiences **high CPU usage and slow response times**.

Solution Using Profiling Tools:

1. **Use top and htop** to identify CPU-intensive processes.

ISDM-NXT

2. **Use perf stat -p <PID>** to analyze CPU performance.
3. **Run valgrind** to check for memory leaks.
4. **Monitor network traffic with iftop** to detect slow API calls.

Outcome:

- **Detected an inefficient loop consuming 80% of CPU cycles.**
- **Fixed memory leak reducing RAM usage by 50%.**
- **Optimized network requests improving response times.**

CHAPTER 9: EXERCISE

1. **Use perf to analyze CPU usage of a running process.**
2. **Run valgrind to check for memory leaks in an application.**
3. **Monitor disk I/O using iotop and iostat.**
4. **Capture network traffic using tcpdump for troubleshooting.**
5. **Trace system calls of an application using strace.**

CONCLUSION

Profiling Linux applications is **crucial for optimizing performance, reducing resource consumption, and troubleshooting issues**. By mastering tools such as **perf, valgrind, iotop, iftop, strace, and ltrace**, developers and system administrators can **ensure efficient, stable, and scalable applications**.

ISDM-NXT

FINAL PROJECT SELECTION & IMPLEMENTATION

CHAPTER 1: INTRODUCTION TO THE FINAL PROJECT

What is the Final Project?

The **Final Project** serves as the **culmination of the skills and knowledge** acquired throughout the Linux course. It involves selecting a **real-world use case**, designing a solution, implementing the necessary configurations, and ensuring **optimal performance, security, and scalability**.

Why is the Final Project Important?

- **Applies theoretical knowledge** to practical scenarios.
- **Enhances troubleshooting and problem-solving skills**.
- **Prepares students for real-world job roles in DevOps, Cloud, and System Administration**.
- **Encourages research, documentation, and best practices in Linux implementation**.

Example Project Scenario

A **startup** wants to deploy a **secure, load-balanced web application** using **Linux servers, Docker, Nginx, and Ansible** to ensure **high availability, performance tuning, and automation**.

CHAPTER 2: CHOOSING THE RIGHT PROJECT

1. Key Factors to Consider

- **Interest and Career Goals**: Choose a project aligned with **cloud computing, DevOps, security, or system administration**.

ISDM-NXT

- **Complexity**: Ensure the project **challenges problem-solving skills** while remaining achievable.
- **Practicality**: Focus on projects that **solve real-world IT challenges**.

2. Example Final Project Ideas

Project Title	Description	Skills Covered
Deploying a Scalable Web Server	Set up a load-balanced Nginx web server with HAProxy and Docker.	Networking, Load Balancing, Web Servers
Automating Server Configuration with Ansible	Configure multiple Linux servers with Ansible for software deployment.	Automation, Configuration Management
Implementing a CI/CD Pipeline with Jenkins	Automate application deployment using Git, Jenkins, and Docker.	DevOps, CI/CD, Containerization
Building a Secure Linux Firewall	Implement iptables and ufw rules to harden server security.	Security, Networking
Setting Up Kubernetes on Linux	Deploy and manage Kubernetes clusters for container orchestration.	Cloud Computing, Kubernetes

3. Selecting a Project

- **Assess feasibility** based on available resources.
- **Ensure project aligns with course objectives**.
- **Consider future scalability and enhancements**.

ISDM-NXT

CHAPTER 3: PLANNING THE PROJECT IMPLEMENTATION

1. Define Project Scope and Requirements

Clearly outline the **goal, expected outcomes, and technology stack**.

Example for "Deploying a Scalable Web Server":

- **Goal:** Set up an Nginx web server with HAProxy for load balancing.
- **Technology:** Linux, Nginx, HAProxy, Docker.
- **Outcome:** Users access the web application via a single entry point, with traffic distributed across multiple backend servers.

2. Set Up a Project Plan

Task	Description	Timeline
Research & Design	Identify best tools, define architecture.	Week 1
Environment Setup	Install OS, configure networking, install dependencies.	Week 2
Implementation	Configure servers, write automation scripts.	Week 3-4
Testing & Debugging	Performance testing, load balancing verification.	Week 5
Final Documentation & Presentation	Report findings, project demo.	Week 6

CHAPTER 4: PROJECT IMPLEMENTATION – STEP-BY-STEP GUIDE

ISDM-NXT

1. Setting Up the Linux Environment

For a **web server deployment project,** start by setting up Linux-based virtual machines or cloud instances:

sudo apt update && sudo apt upgrade -y

sudo apt install nginx haproxy docker.io -y

2. Configuring Load Balancing with HAProxy

Edit the HAProxy configuration file:

sudo nano /etc/haproxy/haproxy.cfg

Add backend server configuration:

```
frontend http_front
   bind *:80
   default_backend web_servers


backend web_servers
   balance roundrobin
   server server1 192.168.1.101:80 check
   server server2 192.168.1.102:80 check
```

Restart HAProxy:

sudo systemctl restart haproxy

3. Automating Deployment with Ansible

Create an **Ansible Playbook** to install and configure Nginx:

```
---
- name: Install and Configure Nginx
```

ISDM-NXT

```
hosts: web_servers
become: yes
tasks:
 - name: Install Nginx
   apt:
     name: nginx
     state: present
 - name: Start Nginx
   service:
     name: nginx
     state: started
```

Run the playbook:

```
ansible-playbook -i inventory.ini nginx_setup.yml
```

4. Monitoring System Performance

Install htop and iotop for monitoring:

```
sudo apt install htop iotop -y
htop
```

This helps analyze **CPU, memory, and disk I/O performance**.

5. Security Hardening

Implement **firewall rules** to secure access:

```
sudo ufw allow OpenSSH
sudo ufw allow 80/tcp
```

ISDM-NXT

sudo ufw enable

CHAPTER 5: TESTING & DEBUGGING THE IMPLEMENTATION

1. Verifying Load Balancing

Use curl to simulate traffic requests:

curl http://<load_balancer_IP>

Each request should route to different backend servers.

2. Stress Testing the System

Install and run Apache Benchmark to test performance:

sudo apt install apache2-utils -y

ab -n 1000 -c 50 http://<load_balancer_IP>/

This helps analyze how the system handles **high traffic loads**.

3. Debugging Common Issues

- **Check server logs for errors:**
- sudo tail -f /var/log/nginx/error.log
- **Verify HAProxy status:**
- sudo systemctl status haproxy

CHAPTER 6: DOCUMENTING THE FINAL PROJECT

1. Writing a Detailed Report

A **final project report** should include:

- **Introduction & Objective**

ISDM-NXT

- **Technology Stack Used**
- **Implementation Steps**
- **Performance Analysis**
- **Challenges & Solutions**
- **Conclusion & Future Improvements**

2. Creating a Project Presentation

Prepare a **PowerPoint presentation** for project demonstration. Include:

- **System Architecture Diagram**
- **Screenshots of Configurations & Tests**
- **Performance Metrics & Graphs**
- **Live Demonstration**

CHAPTER 7: CASE STUDY – REAL-WORLD APPLICATION OF PROJECT IMPLEMENTATION

Scenario:

A **financial institution** needs a **highly available, secure web application** for online transactions.

Solution Using Linux and Load Balancing:

1. **Deploy Nginx web servers across multiple Linux machines.**
2. **Use HAProxy to distribute traffic dynamically.**
3. **Automate server configurations using Ansible.**
4. **Implement firewall and security hardening techniques.**

ISDM-NXT

5. **Monitor performance using htop and stress test with Apache Benchmark**.

Outcome:

- **Increased server uptime from 95% to 99.9%.**
- **Improved response time under heavy traffic loads.**
- **Seamless failover and security enhancements.**

CHAPTER 8: EXERCISE

1. **Set up a Linux server and install a web application.**
2. **Configure a load balancer (HAProxy or Nginx) for scalability.**
3. **Automate the deployment using Ansible.**
4. **Perform a stress test to measure performance.**
5. **Document the implementation process with screenshots and configurations.**

CONCLUSION

The **Final Project** is an opportunity to **apply Linux skills in a real-world setting**, integrating **system administration, automation, security, and performance optimization**. Completing this project prepares students for **DevOps, Cloud, and IT Administration roles**, providing practical experience in **deploying scalable and secure Linux solutions**.

ISDM-NXT

ISDM-NXT

TROUBLESHOOTING AND DEBUGGING LINUX ISSUES

CHAPTER 1: INTRODUCTION TO TROUBLESHOOTING AND DEBUGGING IN LINUX

What is Troubleshooting in Linux?

Troubleshooting in Linux is the **systematic process of identifying, diagnosing, and resolving issues** that impact system performance, application functionality, or security. It involves analyzing logs, monitoring resource utilization, and debugging software and network failures.

Why is Troubleshooting Important?

- **Ensures system stability** by resolving software or hardware failures.
- **Improves performance** by detecting bottlenecks and fixing inefficiencies.
- **Enhances security** by identifying vulnerabilities and suspicious activity.
- **Prevents downtime** by proactively monitoring and fixing errors.

Example Scenario

A **web server running on Linux** crashes frequently, causing downtime. Troubleshooting involves **checking system logs, monitoring CPU/memory usage, and debugging Apache/Nginx errors** to resolve the issue.

ISDM-NXT

CHAPTER 2: SYSTEM LOGS AND DIAGNOSTICS

1. Understanding Linux Logs

Linux maintains various log files that store system and application events.

Log File	Location	Purpose
System Logs	/var/log/syslog or /var/log/messages	Stores general system events.
Kernel Logs	/var/log/kern.log	Contains kernel-related events.
Authentication Logs	/var/log/auth.log	Logs SSH login attempts and sudo access.
Daemon Logs	/var/log/daemon.log	Logs service-related events.
Application Logs	/var/log/apache2/ or /var/log/nginx/	Stores web server logs.

2. Viewing Logs Using journalctl and dmesg

- View recent logs:
- journalctl -xe
- Check logs for a specific service:
- journalctl -u apache2 --no-pager
- View kernel logs:
- dmesg | tail -20

ISDM-NXT

- Filter logs for errors or warnings:
- journalctl | grep -i "error"

CHAPTER 3: CPU AND MEMORY TROUBLESHOOTING

1. Monitoring CPU Usage

Use top or htop to analyze **high CPU-consuming processes**:

top -o %CPU

To find processes consuming the most CPU:

ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%cpu | head

2. Managing High CPU Usage

- Lower the priority of a high-CPU process:
- renice +10 -p <PID>
- Kill an unresponsive process:
- kill -9 <PID>

3. Checking Memory Usage

- View system memory usage:
- free -m
- List memory-consuming processes:
- ps aux --sort=-%mem | head -10
- Check for memory leaks using valgrind:
- valgrind --leak-check=full ./application

ISDM-NXT

4. Resolving Out of Memory (OOM) Issues

If a system runs out of memory, the **OOM killer** terminates processes. To check OOM logs:

dmesg | grep -i "oom"

To adjust swap usage:

sysctl -w vm.swappiness=10

CHAPTER 4: DISK AND FILE SYSTEM ISSUES

1. Checking Disk Usage

- View available disk space:
- df -h
- Find large files consuming space:
- du -sh /var/*
- Identify disk I/O issues using iotop:
- sudo iotop

2. Repairing File System Errors

To check and repair disk errors:

sudo fsck -y /dev/sda1

3. Monitoring Disk Performance

To check read/write performance:

iostat -x 5

ISDM-NXT

CHAPTER 5: NETWORK TROUBLESHOOTING

1. Checking Network Connectivity

- Verify network interfaces:
- ip a
- Test internet connectivity:
- ping -c 5 google.com
- Check routing table:
- ip route show

2. Debugging Network Issues Using netstat and ss

- View active connections:
- netstat -tulnp
- Check which process is using a port:
- ss -tulnp

3. Capturing Network Packets Using tcpdump

To capture HTTP traffic:

sudo tcpdump -i etho port 80

4. Fixing Slow Network Performance

- Restart networking service:
- sudo systemctl restart networking
- Flush DNS cache:

ISDM-NXT

- sudo systemd-resolve --flush-caches

CHAPTER 6: APPLICATION AND SERVICE DEBUGGING

1. Restarting and Checking Services

- Check service status:
- systemctl status apache2
- Restart a service:
- sudo systemctl restart apache2

2. Debugging Service Failures

- Check logs for errors:
- journalctl -u apache2 --no-pager | tail -20
- Start a service in debug mode:
- sudo apachectl -X

3. Debugging Application Errors Using strace

- Trace system calls of a running process:
- strace -p <PID>

CHAPTER 7: KERNEL AND BOOT ISSUES

1. Checking Kernel Errors

- View kernel messages:
- dmesg | grep -i "error"

ISDM-NXT

2. Troubleshooting Boot Failures

- Boot into recovery mode and repair files:
- fsck -y /dev/sda1
- Reinstall bootloader:
- sudo grub-install /dev/sda
- sudo update-grub

CHAPTER 8: CASE STUDY – RESOLVING A SERVER PERFORMANCE ISSUE

Scenario:

A **Linux web server** experiences **slow response times** and **random crashes**.

Solution Using Troubleshooting Techniques:

1. **Check logs (journalctl and /var/log/syslog) for errors.**
2. **Monitor CPU and memory usage using htop and free -m.**
3. **Identify slow disk performance using iotop and iostat.**
4. **Analyze network latency using ping and tcpdump.**
5. **Fix filesystem corruption using fsck and clean logs.**

Outcome:

- **Memory leak identified and fixed.**
- **Disk cleanup reduced I/O delays.**
- **Web server response time improved by 70%.**

ISDM-NXT

CHAPTER 9: EXERCISE

1. **Check system logs for error messages and analyze them.**
2. **Monitor CPU and memory usage using top and htop.**
3. **Analyze disk performance using iostat and iotop.**
4. **Use tcpdump to capture network packets and analyze traffic.**
5. **Debug a failed service using journalctl and restart it.**

CONCLUSION

Troubleshooting and debugging Linux issues are **critical skills** for system administrators and DevOps engineers. By using **logs, system monitoring tools, and network analysis utilities**, issues can be **quickly identified and resolved** to ensure **optimal performance and uptime**.

ISDM-NXT

BEST PRACTICES FOR LINUX SYSTEM ADMINISTRATION

CHAPTER 1: INTRODUCTION TO LINUX SYSTEM ADMINISTRATION

What is Linux System Administration?

Linux system administration is the **management, configuration, maintenance, and security** of Linux-based systems to ensure optimal performance and uptime. System administrators (SysAdmins) play a crucial role in **monitoring system health, automating repetitive tasks, securing systems, and troubleshooting issues**.

Why Are Best Practices Important?

- **Ensures system stability and high availability.**
- **Improves security by minimizing vulnerabilities.**
- **Enhances performance through resource optimization.**
- **Automates tasks to reduce human errors.**

Example Scenario

A **company running web applications on Linux servers** needs to ensure **24/7 uptime** and protect against **cyber threats**. By following best practices, they **prevent downtime, automate security updates, and optimize resource utilization**.

CHAPTER 2: USER AND ACCESS MANAGEMENT

1. Creating and Managing Users

ISDM-NXT

- To create a new user:
- sudo adduser username
- To assign a user to the sudo group:
- sudo usermod -aG sudo username
- To list all system users:
- cat /etc/passwd

2. Implementing SSH Key-Based Authentication

- Generate an SSH key pair:
- ssh-keygen -t rsa -b 4096
- Copy the public key to a remote server:
- ssh-copy-id username@remote-server-ip
- Disable password authentication in /etc/ssh/sshd_config:
- PasswordAuthentication no

Restart SSH service:

sudo systemctl restart sshd

3. Implementing Role-Based Access Control (RBAC)

- Restrict access using groups:
- sudo groupadd developers
- sudo usermod -aG developers username
- Use sudo privileges for command execution:
- sudo visudo

ISDM-NXT

Add:

username ALL=(ALL) NOPASSWD: /usr/bin/systemctl restart apache2

CHAPTER 3: SYSTEM MONITORING AND PERFORMANCE TUNING

1. Monitoring CPU, Memory, and Disk Usage

- View real-time CPU and memory usage:
- top
- View memory usage in MB:
- free -m
- Monitor disk space usage:
- df -h

2. Using Advanced Monitoring Tools

- htop (interactive process viewer):
- sudo apt install htop -y
- htop
- iotop (monitor disk I/O activity):
- sudo apt install iotop -y
- sudo iotop

3. Automating Resource Alerts

- Set up automated email alerts for high CPU usage:

ISDM-NXT

- echo 'High CPU Usage Detected' | mail -s 'Alert' admin@example.com
- Use **Prometheus & Grafana** for real-time system monitoring.

Chapter 4: Automating Tasks and System Maintenance

1. Using Cron Jobs for Scheduling Tasks

- Open the crontab editor:
- crontab -e
- Example: Run a backup every day at midnight:
- 0 0 * * * /usr/bin/rsync -av /data /backup/

2. Automating Software Updates

Enable automatic security updates:

sudo apt install unattended-upgrades -y

sudo dpkg-reconfigure unattended-upgrades

3. Automating System Cleanup

- Remove old logs:
- find /var/log -type f -name "*.log" -mtime +30 -exec rm -f {} \;
- Clear unused packages:
- sudo apt autoremove -y

Chapter 5: Backup and Disaster Recovery

ISDM-NXT

1. Creating System Backups with Rsync

To back up /var/www to a remote server:

rsync -avz /var/www/ user@backup-server:/backups/

2. Using Tar for Archiving Files

tar -czvf backup.tar.gz /home/user/documents

3. Setting Up Snapshots with LVM

Create a snapshot of a logical volume:

lvcreate --size 1G --snapshot --name backup_snap /dev/vgo/lv_data

---

CHAPTER 6: SECURITY BEST PRACTICES

1. Hardening SSH Access

- Change the default SSH port:
- sudo nano /etc/ssh/sshd_config

Modify:

Port 2222

Restart SSH:

sudo systemctl restart sshd

2. Configuring a Firewall (UFW)

- Allow only SSH, HTTP, and HTTPS traffic:
- sudo ufw allow OpenSSH
- sudo ufw allow 80/tcp

ISDM-NXT

- sudo ufw allow 443/tcp
- sudo ufw enable

3. Implementing Intrusion Detection with Fail2Ban

- Install Fail2Ban:
- sudo apt install fail2ban -y
- Configure SSH protection:
- sudo nano /etc/fail2ban/jail.local

Add:

[sshd]

enabled = true

bantime = 3600

maxretry = 3

CHAPTER 7: NETWORK CONFIGURATION AND LOAD BALANCING

1. Managing Network Interfaces

- Display network configuration:
- ip a
- Restart the network service:
- sudo systemctl restart networking

2. Configuring Load Balancing with HAProxy

- Install HAProxy:

ISDM-NXT

- sudo apt install haproxy -y
- Configure backend servers:
- sudo nano /etc/haproxy/haproxy.cfg

Add:

```
frontend http_front
  bind *:80
  default_backend web_servers

backend web_servers
  balance roundrobin
  server server1 192.168.1.101:80 check
  server server2 192.168.1.102:80 check
```

- Restart HAProxy:
- sudo systemctl restart haproxy

CHAPTER 8: CASE STUDY – OPTIMIZING A LINUX WEB SERVER FOR HIGH TRAFFIC

Scenario:

A **company's web application** running on Linux experiences **slow load times and high CPU usage**.

Solution Using Best Practices:

1. **Optimize CPU & Memory usage** (htop, free -m).

ISDM-NXT

2. **Configure HAProxy for load balancing** across multiple servers.
3. **Use cron jobs to automate daily log cleanups.**
4. **Harden SSH access by disabling root login and using Fail2Ban.**
5. **Implement backups using Rsync and store copies remotely.**

Outcome:

- **Improved application response time by 50%.**
- **Reduced CPU usage with load balancing.**
- **Secured the system against brute-force attacks.**

CHAPTER 9: EXERCISE

1. **Create a new user and set up SSH key authentication.**
2. **Configure a firewall to allow only HTTP, HTTPS, and SSH access.**
3. **Set up a cron job to clear logs older than 30 days.**
4. **Monitor system performance using htop and iotop.**
5. **Set up an automated backup using rsync.**

CONCLUSION

Linux system administration **requires a combination of security, automation, performance tuning, and network management**. Following best practices **enhances stability, security, and efficiency**, making systems **robust and scalable**.

ISDM-NXT

Assignment Solution: Optimize and Benchmark a Linux System for Performance – Step-by-Step Guide

Objective

This assignment provides a **step-by-step guide** to optimizing and benchmarking a **Linux system** for better performance. It covers **CPU, memory, disk, and network optimizations**, followed by **benchmarking tests** to measure system improvements.

Step 1: System Analysis Before Optimization

1. Check System Resource Usage

Before making optimizations, gather system performance data.

- **Check CPU load and memory usage:**
- top -o %CPU
- free -m
- **Monitor disk performance:**
- iostat -x 5
- **Check network bandwidth:**
- sudo iftop -i etho
- **Check running processes and their resource consumption:**
- ps aux --sort=-%cpu | head -10

Step 2: CPU Performance Optimization

ISDM-NXT

1. Adjust CPU Scheduling for Performance

Change CPU governor to **performance mode**:

sudo cpufreq-set -g performance

2. Optimize Process Scheduling

- Set **higher priority** for critical tasks:
- renice -5 -p <PID>
- Lower priority of **background processes**:
- renice 10 -p <PID>

3. Use Multi-Core Processing Efficiently

To check the number of CPU cores:

nproc

To ensure parallel execution:

taskset -c 0,1 <command>

STEP 3: MEMORY MANAGEMENT OPTIMIZATION

1. Adjust Swappiness for Better RAM Usage

To **reduce swap usage** and prioritize RAM:

sudo sysctl -w vm.swappiness=10

Make it permanent:

echo "vm.swappiness=10" | sudo tee -a /etc/sysctl.conf

2. Clear Cache and Buffers

sync; echo 3 | sudo tee /proc/sys/vm/drop_caches

ISDM-NXT

3. Enable Huge Pages for Memory-Intensive Applications

```
echo "vm.nr_hugepages=128" | sudo tee -a /etc/sysctl.conf
sudo sysctl -p
```

Step 4: Disk Performance Optimization

1. Change I/O Scheduler

Check current scheduler:

```
cat /sys/block/sda/queue/scheduler
```

Set the **deadline** scheduler (best for databases and high-speed disks):

```
echo deadline | sudo tee /sys/block/sda/queue/scheduler
```

2. Enable Write Caching for Faster Disk Writes

```
sudo hdparm -W1 /dev/sda
```

3. Optimize File System Performance

- Enable **writeback mode** for better performance:
- sudo tune2fs -o journal_data_writeback /dev/sda1
- Set file system to **use less reserved space**:
- sudo tune2fs -m 1 /dev/sda1

Step 5: Network Performance Optimization

1. Enable TCP Fast Open for Faster Connections

```
echo "net.ipv4.tcp_fastopen=3" | sudo tee -a /etc/sysctl.conf
sudo sysctl -p
```

ISDM-NXT

2. Increase Network Buffer Sizes

sudo sysctl -w net.core.rmem_max=16777216

sudo sysctl -w net.core.wmem_max=16777216

3. Optimize Connection Tracking for High-Traffic Servers

sudo sysctl -w net.nf_conntrack_max=1048576

Step 6: Benchmarking the Optimized System

1. Benchmark CPU Performance Using sysbench

Install sysbench:

sudo apt install sysbench -y

Run a **CPU stress test**:

sysbench cpu --cpu-max-prime=20000 --threads=4 run

2. Benchmark Memory Performance

sysbench memory --memory-block-size=1M --memory-total-size=10G run

3. Benchmark Disk Read/Write Speed

dd if=/dev/zero of=/tmp/testfile bs=1M count=1000 oflag=direct

4. Benchmark Network Performance Using iperf3

Install iperf3:

sudo apt install iperf3 -y

Run server:

iperf3 -s

Run client:

ISDM-NXT

iperf3 -c <server-ip> -P 10 -t 60

Step 7: Analyze Benchmark Results and Compare Performance

1. Compare Pre-Optimization and Post-Optimization Results

Metric	Before Optimization	After Optimization	Improvement
CPU Benchmark (sysbench)	50,000 events	75,000 events	+50%
Memory Throughput	5GB/sec	8GB/sec	+60%
Disk Write Speed	100MB/sec	250MB/sec	+150%
Network Speed (iperf3)	800 Mbps	1.2 Gbps	+50%

2. Identify Further Areas of Improvement

- If **CPU performance** is still slow → Consider upgrading CPU or optimizing multi-threading.
- If **memory usage is high** → Increase RAM or enable memory compression (zram).
- If **disk performance is slow** → Use **NVMe SSDs** instead of HDDs.
- If **network latency is high** → Optimize firewall rules and use **CDNs** for content delivery.

ISDM-NXT

Case Study – Optimizing a Linux Web Server for High Performance

Scenario:

A **web hosting company** is experiencing **slow page load times and frequent performance issues**.

Solution Using Optimization Techniques:

1. **Reduced CPU load** by optimizing scheduling and enabling **performance governor**.
2. **Improved memory management** by tuning swappiness and enabling huge pages.
3. **Increased disk I/O performance** by switching to the **deadline scheduler** and enabling **write caching**.
4. **Enhanced network throughput** by tuning TCP settings and enabling **TCP Fast Open**.
5. **Verified improvements** using **benchmarking tests** (sysbench, iperf3).

Outcome:

- **Page load times improved by 60%**.
- **Server handled 50% more concurrent requests**.
- **Reduced system downtime and improved user experience**.

STEP 8: EXERCISE

1. **Measure system performance before optimization using sysbench and iostat**.
2. **Optimize CPU performance using process scheduling and governor settings**.

ISDM-NXT

3. **Adjust memory settings (swappiness, cache clearing) and measure improvement**.
4. **Change the I/O scheduler and analyze disk performance using iostat**.
5. **Tweak network performance settings and compare speed before and after optimization**.

CONCLUSION

By following these optimization steps, a Linux system can **run faster, handle higher workloads, and improve overall stability**. Using **benchmarking tools**, administrators can measure performance **before and after optimization** to ensure effective tuning.

ISDM-NXT

Assignment Solution: Complete a Final Project on a Real-World Linux Use Case – Step-by-Step Guide

Objective

The final project aims to apply **real-world Linux administration, automation, security, and performance tuning** to a **practical use case**. This step-by-step guide will demonstrate **deploying a secure, high-performance Linux web server with load balancing, automation, and monitoring**.

Step 1: Define the Real-World Linux Use Case

Project Title: Deploying a Scalable and Secure Web Server on Linux

Project Scope:

- Deploy **multiple web servers** on Linux.
- Implement **load balancing** for high availability.
- Automate configuration using **Ansible**.
- Secure the system using **firewalls, SSH hardening, and Fail2Ban**.
- Monitor system performance using **Prometheus and Grafana**.

Technology Stack:

- **Linux Distribution:** Ubuntu 22.04
- **Web Server:** Nginx

ISDM-NXT

- **Load Balancer:** HAProxy
- **Automation Tool:** Ansible
- **Monitoring Tools:** Prometheus & Grafana
- **Security Tools:** UFW Firewall & Fail2Ban

Step 2: Setting Up the Linux Environment

1. Create Virtual Machines or Cloud Instances

Deploy three Linux instances:

- **Load Balancer (HAProxy)**
- **Web Server 1 (Nginx)**
- **Web Server 2 (Nginx)**

2. Update and Secure the Linux Servers

sudo apt update && sudo apt upgrade -y

sudo ufw allow OpenSSH

sudo ufw enable

3. Configure SSH Key-Based Authentication

On your local machine:

ssh-keygen -t rsa -b 4096

Copy the public key to all servers:

ssh-copy-id user@server-ip

ISDM-NXT

Step 3: Deploy Nginx on Web Servers

1. Install Nginx

On **Web Server 1 and Web Server 2**:

sudo apt install nginx -y

2. Configure a Simple Web Page

Modify the Nginx default index page:

echo "<h1>Web Server 1 Running</h1>" | sudo tee /var/www/html/index.html

For Web Server 2, modify the content to:

echo "<h1>Web Server 2 Running</h1>" | sudo tee /var/www/html/index.html

3. Start and Enable Nginx

sudo systemctl start nginx

sudo systemctl enable nginx

Step 4: Configure HAProxy as a Load Balancer

1. Install HAProxy on Load Balancer Server

sudo apt install haproxy -y

2. Edit HAProxy Configuration File

sudo nano /etc/haproxy/haproxy.cfg

Add the following configuration:

frontend http_front

ISDM-NXT

```
    bind *:80
    default_backend web_servers

backend web_servers
    balance roundrobin
    server server1 192.168.1.101:80 check
    server server2 192.168.1.102:80 check
```

3. Restart HAProxy to Apply Changes

```
sudo systemctl restart haproxy
```

4. Test Load Balancing

Access the HAProxy public IP in a browser:

http://<load_balancer_IP>

The page should alternate between **Web Server 1** and **Web Server 2**.

STEP 5: AUTOMATE DEPLOYMENT USING ANSIBLE

1. Install Ansible on the Control Node

```
sudo apt install ansible -y
```

2. Configure Ansible Inventory File

```
sudo nano /etc/ansible/hosts
```

Add the following:

ISDM-NXT

[web_servers]

192.168.1.101

192.168.1.102

[load_balancer]

192.168.1.100

3. Create an Ansible Playbook for Nginx Deployment

nano deploy_nginx.yml

Add the following:

```
---
- name: Install and Configure Nginx
 hosts: web_servers
 become: yes
 tasks:
  - name: Install Nginx
   apt:
    name: nginx
    state: present

  - name: Start Nginx
   service:
```

ISDM-NXT

name: nginx

state: started

4. Run the Ansible Playbook

ansible-playbook -i /etc/ansible/hosts deploy_nginx.yml

STEP 6: SECURE THE LINUX WEB SERVER

1. Configure UFW Firewall

sudo ufw allow 80/tcp

sudo ufw allow 443/tcp

sudo ufw enable

2. Install and Configure Fail2Ban

sudo apt install fail2ban -y

sudo nano /etc/fail2ban/jail.local

Add:

[sshd]

enabled = true

bantime = 3600

maxretry = 3

Restart Fail2Ban:

sudo systemctl restart fail2ban

ISDM-NXT

Step 7: Implement System Monitoring Using Prometheus and Grafana

1. Install Prometheus on Load Balancer Server

sudo apt install prometheus -y

2. Configure Prometheus to Monitor Web Servers

Edit Prometheus configuration file:

sudo nano /etc/prometheus/prometheus.yml

Add:

```
scrape_configs:
 - job_name: 'web_servers'
   static_configs:
    - targets: ['192.168.1.101:80', '192.168.1.102:80']
```

Restart Prometheus:

sudo systemctl restart prometheus

3. Install Grafana for Visualization

sudo apt install grafana -y

sudo systemctl start grafana

sudo systemctl enable grafana

Access Grafana in a browser:

http://<load_balancer_IP>:3000

Default credentials:

ISDM-NXT

- Username: admin
- Password: admin

Step 8: Performance Testing

1. Load Testing Using Apache Benchmark

ab -n 1000 -c 50 http://<load_balancer_IP>/

- -n 1000 ⟶ Send 1000 requests
- -c 50 ⟶ 50 concurrent users

2. Analyze Logs for Errors

journalctl -u nginx --no-pager | tail -20

Case Study: Optimizing a Linux-Based E-Commerce Website

Scenario:

A company running an **e-commerce website** experiences **slow load times and high downtime during traffic spikes**.

Solution:

1. **Implemented load balancing** with HAProxy for high availability.
2. **Automated server deployment** using Ansible.
3. **Hardened security** with firewalls and Fail2Ban.
4. **Monitored performance** using Prometheus and Grafana.

Outcome:

ISDM-NXT

- **Improved uptime from 95% to 99.9%.**
- **Increased concurrent users handling by 60%.**
- **Reduced security incidents by 80%.**

Step 9: Exercise

1. **Deploy a web server using Ansible.**
2. **Configure HAProxy for load balancing.**
3. **Harden SSH security using Fail2Ban.**
4. **Set up Prometheus to monitor system performance.**
5. **Conduct load testing using Apache Benchmark.**

Conclusion

This **real-world Linux project** integrates **deployment, automation, security, and monitoring** to create a **scalable, high-performance** web server. Completing this project provides hands-on experience for **DevOps, Cloud, and System Administration** roles.

ISDM-NXT