Independent Skill Development Mission

Independent Skill Development Mission NXT
EMPOWERING FUTURES, SHAPING TOMORROW

ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

WORKING ON A LIVE UNIX SERVER

CHAPTER 1: INTRODUCTION TO LIVE UNIX SERVER ADMINISTRATION

What is a Live UNIX Server?

A **live UNIX server** is an actively running system that provides **real-time services** to users and applications. These services include **web hosting, databases, networking, security monitoring, and software development environments**. Unlike local development machines, working on a live server requires **careful execution of commands** to prevent unintended disruptions or system failures.

Key Responsibilities of UNIX Server Administrators

- **Managing system resources (CPU, memory, and disk usage).**
- **Monitoring active processes and services.**
- **Configuring and maintaining security policies.**
- **Performing system updates without downtime.**
- **Handling user and file permissions.**
- **Ensuring backup and disaster recovery procedures.**

Precautions When Working on a Live Server

- Always **verify commands before execution** to avoid accidental deletions.
- Use the screen or tmux command to avoid session disconnection.
- Perform **backups before making major system changes**.
- Monitor **real-time system logs** while making changes.
- Use **restricted user privileges** when executing non-administrative tasks.

Example: Checking System Uptime and Active Users

To check how long the server has been running and who is logged in:

uptime

who

Exercise

1. Use uptime to check the system load and running time.
2. Run who to list currently logged-in users.

Case Study: Preventing Downtime in a Cloud Hosting Environment

A cloud hosting provider experienced **unexpected server crashes** due to **uncontrolled CPU usage** by user applications. By implementing **real-time monitoring tools** and **setting resource limits,** the team was able to **prevent overloads and reduce downtime by 60%**.

CHAPTER 2: MONITORING AND MANAGING SYSTEM RESOURCES ON A LIVE SERVER

1. Checking CPU and Memory Usage

A live UNIX server must always have **sufficient CPU and memory resources** to handle running applications efficiently. Overloaded resources can lead to **server crashes, slowdowns, or failed services**.

A. Monitoring CPU Usage

The top command provides **real-time system monitoring**:

top -o %CPU

To find the most **CPU-intensive processes**, use:

ps -eo pid,ppid,cmd,%cpu --sort=-%cpu | head -10

B. Monitoring Memory Usage

To check memory consumption:

free -m

To find **memory-hungry processes**:

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

Exercise

1. Identify the **top CPU-consuming process** and lower its priority using renice.
2. Check **memory usage** and free up unused caches using sync; echo 3 | sudo tee /proc/sys/vm/drop_caches.

Case Study: Reducing Server Slowdowns in a Database System

A financial company experienced **delays in transaction processing** due to **high memory consumption**. By tuning **database memory settings** and configuring **swappiness,** they improved system performance by **30%**.

Chapter 3: Managing User Access and Permissions on a Live Server

1. Managing User Accounts

A live server **hosts multiple users** with different levels of access. It is crucial to **restrict unauthorized access** and enforce **the principle of least privilege (PoLP)**.

A. Listing Active Users

To list logged-in users:

who

To check user activity:

w

B. Adding and Deleting Users

To create a new user:

sudo useradd -m -s /bin/bash username

sudo passwd username

To remove a user:

sudo userdel -r username

2. Managing File Permissions

File permissions prevent unauthorized access to system files.

To check permissions:

ls -l /home/user/file.txt

To modify permissions:

chmod 640 /home/user/file.txt

To set ownership:

chown user:group /home/user/file.txt

Exercise

1. Create a new user and assign specific file permissions.
2. Use who and w to monitor user activity.

Case Study: Securing Confidential Data in an Enterprise Server

An IT company faced **unauthorized file access** on its servers. By **implementing proper user permissions and access control lists (ACLs),** they prevented **data leaks** and strengthened security policies.

CHAPTER 4: MANAGING PROCESSES AND SERVICES ON A LIVE SERVER

1. Managing System Processes

A live UNIX server runs **multiple background processes**. Administrators must ensure **critical services remain active** while terminating unnecessary ones.

A. Listing Active Processes

ps aux | less

To find a specific process:

ps aux | grep apache

B. Stopping and Restarting Services

To stop a service:

sudo systemctl stop apache2

To restart a service:

sudo systemctl restart apache2

Exercise

1. Find all processes related to ssh and restart the service.
2. List all running services and identify any failed ones.

Case Study: Restarting Critical Services in a Production Web Server

An e-commerce website faced **unexpected downtime** due to a failed Apache service. By implementing **process monitoring with systemd,** administrators set up **automatic service restarts** to prevent future outages.

CHAPTER 5: MANAGING LOGS AND TROUBLESHOOTING ISSUES ON A LIVE SERVER

1. Checking System Logs

Log files are **crucial** for troubleshooting issues and monitoring system activity.

A. Viewing System Logs

To view real-time logs:

sudo journalctl -f

To check authentication logs:

sudo cat /var/log/auth.log

B. Finding Errors in Logs

To find errors related to a service (e.g., Apache):

sudo journalctl -u apache2 --since "1 hour ago"

2. Handling System Failures

If a system crashes, restart services and check logs:

sudo systemctl restart networking

sudo journalctl -xe

Exercise

1. View the last **10 failed SSH login attempts** using sudo cat /var/log/auth.log | grep "Failed password" | tail -10.
2. Restart a service and verify logs for any errors.

Case Study: Resolving Network Downtime in a Cloud Server

A cloud server **lost network connectivity** due to incorrect firewall rules. By analyzing **system logs and troubleshooting firewall settings,** the administrator **restored connectivity within 10 minutes**.

CONCLUSION

This guide covered:

✅ **Monitoring system performance (CPU, memory, disk, network).**

✅ **Managing user access, permissions, and active services.**

✅ **Troubleshooting system logs and resolving live server issues.**

✅ **Implementing best practices for secure and efficient UNIX server management.**

DEPLOYING WEB APPLICATIONS ON UNIX

CHAPTER 1: INTRODUCTION TO WEB APPLICATION DEPLOYMENT ON UNIX

What is Web Application Deployment?

Web application deployment refers to **the process of setting up, configuring, and running a web-based application on a UNIX server**. This includes **installing necessary software, configuring the server, setting up databases, handling security, and ensuring availability**.

Why Deploy Web Applications on UNIX?

- **Stability and Performance** – UNIX-based servers (Linux, BSD) are highly reliable.
- **Security** – UNIX offers robust security features like **file permissions, firewalls, and SELinux**.
- **Flexibility** – Support for various programming languages (Python, PHP, Java, Node.js, etc.).
- **Scalability** – Can handle high traffic loads with **load balancing and clustering**.

Common Web Server Options on UNIX

Web Server	Description	Use Case
Apache	Most widely used, flexible	General-purpose hosting
Nginx	High-performance, efficient	Static content, reverse proxy

Lighttpd	Lightweight web server	Embedded systems, low-resource VPS
Tomcat	Java-based application server	Java applications (JSP, Servlets)

Example: Checking System Information Before Deployment

To check system details before deploying an application:

uname -a

df -h

free -m

Exercise

1. Run uname -a to check your UNIX system details.
2. Check available disk space using df -h.

Case Study: Deploying a High-Traffic E-Commerce Website

A company launched an **e-commerce platform** using **Nginx, PHP, and MySQL on UNIX servers**. By optimizing caching and load balancing, they improved **site speed by 50%** and handled **high user traffic efficiently**.

CHAPTER 2: SETTING UP A WEB SERVER ON UNIX

1. Installing a Web Server (Apache/Nginx)

A. Installing Apache (Ubuntu/Debian)

sudo apt update

sudo apt install apache2 -y

Start and enable the service:

sudo systemctl start apache2

sudo systemctl enable apache2

Check if Apache is running:

sudo systemctl status apache2

Verify installation by visiting:

http://server-ip

B. Installing Nginx (CentOS/RHEL)

sudo yum install epel-release -y

sudo yum install nginx -y

Start and enable Nginx:

sudo systemctl start nginx

sudo systemctl enable nginx

2. Configuring the Web Server

For Apache, modify the **default configuration file**:

sudo nano /etc/apache2/sites-available/000-default.conf

For Nginx, modify the **server block**:

sudo nano /etc/nginx/nginx.conf

Restart the service to apply changes:

sudo systemctl restart apache2 # or nginx

Exercise

1. Install and start **Apache or Nginx** on your UNIX server.
2. Modify the default configuration and restart the web server.

Case Study: Choosing the Right Web Server for a Media Streaming Platform

A media company compared **Apache vs. Nginx** for video streaming. They deployed **Nginx due to its efficient handling of static content,** reducing **server response time by 40%**.

CHAPTER 3: DEPLOYING A WEB APPLICATION (PHP, PYTHON, NODE.JS)

1. Deploying a PHP-Based Web Application

A. Installing PHP and Required Modules

sudo apt install php libapache2-mod-php php-mysql -y

Verify PHP installation:

php -v

B. Deploying a Sample PHP App

1. Navigate to the web root directory:
2. cd /var/www/html
3. Create a simple PHP file:
4. echo "<?php phpinfo(); ?>" | sudo tee /var/www/html/info.php
5. Restart Apache:

6. sudo systemctl restart apache2
7. Open http://server-ip/info.php in a browser to verify.

2. Deploying a Python Flask Web Application

A. Install Python and Flask

sudo apt install python3-pip -y

pip3 install flask

B. Create a Flask App

mkdir ~/flaskapp && cd ~/flaskapp

nano app.py

Paste the following:

```
from flask import Flask

app = Flask(__name__)


@app.route('/')

def home():

   return "Hello, World!"


if __name__ == '__main__':

   app.run(host='0.0.0.0', port=5000)
```

Run the Flask app:

python3 app.py

Visit: http://server-ip:5000

3. Deploying a Node.js Web Application

A. Install Node.js

sudo apt install nodejs npm -y

B. Create a Node.js App

mkdir ~/nodeapp && cd ~/nodeapp

nano server.js

Paste:

```
const http = require('http');
const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello, Node.js!');
});
server.listen(3000, () => console.log('Server running on port 3000'));
```

Run the app:

node server.js

Visit: http://server-ip:3000

Exercise

1. Deploy a simple **PHP application** and verify using a web browser.
2. Set up a **Flask or Node.js application** and test it locally.

Case Study: Deploying a Python Flask API for a Machine Learning Model

A startup built an **AI-based recommendation system** using Python. They deployed a **Flask API on an Nginx server,** reducing API response time by **30%** compared to traditional RESTful architectures.

CHAPTER 4: CONFIGURING SECURITY AND PERFORMANCE OPTIMIZATION

1. Securing the Web Server

A. Setting Up a Firewall

```
sudo ufw allow 'Apache Full'  # For Apache
sudo ufw allow 'Nginx Full'  # For Nginx
sudo ufw enable
```

B. Enabling SSL with Let's Encrypt

```
sudo apt install certbot python3-certbot-apache -y
sudo certbot --apache -d yourdomain.com
```

For Nginx:

```
sudo certbot --nginx -d yourdomain.com
```

2. Improving Performance with Caching

Enable caching in Nginx:

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

Add:

fastcgi_cache_path /var/cache/nginx levels=1:2 keys_zone=FASTCGI:100m inactive=60m;

Restart Nginx:

sudo systemctl restart nginx

Exercise

1. Configure a **firewall rule** to allow only web traffic.
2. Enable **SSL using Let's Encrypt** for a secure HTTPS website.

Case Study: Improving Performance for a Large-Scale Blog

A blogging platform **enabled caching and SSL on Nginx,** improving load times by **50%** and securing data transmission against cyber threats.

CONCLUSION

This guide covered:

✅ **Installing and configuring web servers (Apache, Nginx).**
✅ **Deploying PHP, Python, and Node.js applications.**
✅ **Securing and optimizing UNIX-based web applications.**
✅ **Automating deployments for better efficiency.**

SETTING UP VIRTUALIZATION AND CONTAINERS (DOCKER) IN UNIX/LINUX

CHAPTER 1: INTRODUCTION TO VIRTUALIZATION AND CONTAINERS

What is Virtualization?

Virtualization is a technology that allows you to **run multiple operating systems on a single physical server** using virtual machines (VMs). It provides **isolation, resource efficiency, and ease of deployment** for different applications.

What are Containers?

Containers are **lightweight, portable, and self-sufficient environments** that include all the necessary dependencies to run an application. Unlike traditional virtualization, **containers share the host OS kernel**, making them **faster and more efficient** than VMs.

Difference Between Virtual Machines and Containers

Feature	Virtual Machines (VMs)	Containers
Size	Large (GBs)	Small (MBs)
Performance	Slow (requires full OS)	Fast (shares host OS)
Isolation	Full OS-level isolation	Process-level isolation
Startup Time	Minutes	Seconds
Use Case	Running multiple OS instances	Running lightweight microservices

Popular Virtualization and Container Technologies

- **Virtualization:** VMware, VirtualBox, KVM, Xen
- **Containers:** Docker, Podman, Kubernetes, LXC

Example: Checking System Virtualization Support

To check if your system supports virtualization:

egrep -c '(vmx|svm)' /proc/cpuinfo

- If the output is 1 or higher, virtualization is supported.

Exercise

1. Check if your system supports virtualization using the above command.
2. List running virtual machines using virsh list --all.

Case Study: Migrating from Virtual Machines to Containers

A financial institution used **VMs for hosting microservices**, but experienced **slow performance and high resource consumption**. By migrating to **Docker containers**, they **reduced infrastructure costs by 40%** while improving application speed.

CHAPTER 2: SETTING UP VIRTUALIZATION ON UNIX/LINUX

1. Installing and Configuring KVM (Kernel-based Virtual Machine)

KVM is a popular **open-source virtualization solution** built into the Linux kernel.

A. Installing KVM on Ubuntu/Debian

sudo apt update

sudo apt install -y qemu-kvm libvirt-daemon-system libvirt-clients bridge-utils virt-manager

B. Installing KVM on RHEL/CentOS

sudo yum install -y qemu-kvm libvirt virt-install bridge-utils

C. Enabling and Starting the KVM Service

sudo systemctl enable libvirtd

sudo systemctl start libvirtd

2. Creating a Virtual Machine Using virt-install

```
sudo virt-install \
--name ubuntu_vm \
--ram 2048 \
--disk path=/var/lib/libvirt/images/ubuntu.qcow2,size=20 \
--vcpus 2 \
--os-type linux \
--os-variant ubuntu20.04 \
--network bridge=virbro \
--graphics none \
--console pty,target_type=serial \
--cdrom /path/to/ubuntu.iso
```

This command creates a **new virtual machine** with 2GB RAM, 20GB disk, and Ubuntu 20.04 ISO.

Exercise

1. Install KVM on your UNIX system.
2. Create a virtual machine using virt-install.

Case Study: Running Multiple Virtual Machines for Software Testing

A software company **uses KVM to create isolated test environments** for different Linux distributions. This setup allows **secure testing without affecting production systems**.

CHAPTER 3: SETTING UP DOCKER FOR CONTAINERIZATION

1. Installing Docker on UNIX/Linux

Docker is the most widely used **containerization platform** that allows **efficient and fast application deployment**.

A. Installing Docker on Ubuntu/Debian

sudo apt update

sudo apt install -y docker.io

B. Installing Docker on RHEL/CentOS

sudo yum install -y yum-utils

sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

sudo yum install -y docker-ce docker-ce-cli containerd.io

C. Starting Docker Service

sudo systemctl start docker

sudo systemctl enable docker

2. Verifying Docker Installation

sudo docker --version

sudo docker run hello-world

This runs a **test container** to verify Docker is working.

Exercise

1. Install Docker on your UNIX system.
2. Run the hello-world container to verify installation.

Case Study: Deploying a Web Application with Docker

An e-commerce company **migrated from traditional hosting to Docker containers,** reducing deployment time from **hours to minutes** while improving application portability.

CHAPTER 4: WORKING WITH DOCKER CONTAINERS

1. Running a Container

To run an **Nginx web server** container:

sudo docker run -d -p 80:80 --name webserver nginx

- -d ⟶ Runs the container in **detached mode**.
- -p 80:80 ⟶ Maps container port 80 to host port 80.
- --name webserver ⟶ Assigns a name to the container.

2. Listing Running Containers

sudo docker ps

To list **all containers,** including stopped ones:

sudo docker ps -a

3. Stopping and Removing a Container

sudo docker stop webserver

sudo docker rm webserver

4. Pulling and Running Custom Images

To pull an **Ubuntu container image**:

sudo docker pull ubuntu

To start an interactive Ubuntu container:

sudo docker run -it ubuntu bash

Exercise

1. Run an **Nginx container** and access it in a web browser.
2. Start an **Ubuntu container** and explore its environment.

Case Study: Deploying Scalable Applications Using Docker

A social media startup **moved from traditional servers to Docker containers,** enabling **rapid scaling and automated deployments,** reducing downtime and infrastructure costs.

CHAPTER 5: MANAGING DOCKER IMAGES AND VOLUMES

1. Building a Custom Docker Image

Create a Dockerfile:

nano Dockerfile

Add the following content:

```
FROM ubuntu:latest

RUN apt update && apt install -y apache2

CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

Build the image:

```
sudo docker build -t custom-apache .
```

Run the container:

```
sudo docker run -d -p 8080:80 custom-apache
```

2. Using Docker Volumes for Data Persistence

Create a volume:

```
sudo docker volume create mydata
```

Run a container with persistent storage:

```
sudo docker run -d -v mydata:/var/lib/mysql --name db mysql
```

Exercise

1. Create a **custom Docker image** and run a container from it.
2. Use **Docker volumes** to persist application data.

Case Study: Using Docker Volumes for Database Backup

A company **uses Docker volumes to store MySQL database data,** ensuring that **data is not lost when the container is restarted**.

CONCLUSION

This guide covered:

✅ **Setting up virtualization using KVM on UNIX/Linux.**

✅ **Installing Docker and running containerized applications.**

✅ **Deploying and managing containers using docker run, docker ps, and docker volume.**

✅ **Building custom Docker images and persisting data with volumes.**

CLOUD COM

CLOUD COMPUTING WITH UNIX (AWS, AZURE)

CHAPTER 1: INTRODUCTION TO CLOUD COMPUTING WITH UNIX

What is Cloud Computing?

Cloud computing is a technology that allows users to **access computing resources (servers, storage, networking, and software) over the internet** rather than owning and maintaining physical infrastructure. It provides **scalability, reliability, and cost efficiency**, making it a preferred choice for businesses and developers.

UNIX-based systems (Linux, BSD, and UNIX distributions) are widely used in **cloud environments** because of their **stability, security, and flexibility**. Most cloud providers, including **AWS (Amazon Web Services) and Azure (Microsoft Azure), offer UNIX-based virtual machines, containers, and managed services**.

Key Benefits of Cloud Computing with UNIX

- **Scalability** – Easily scale resources up or down based on demand.
- **Cost Efficiency** – Pay only for what you use, reducing infrastructure costs.
- **Security** – Built-in firewalls, encryption, and access control ensure system protection.
- **Automation** – Use **shell scripting, Terraform, and Ansible** to automate infrastructure.
- **Global Reach** – Deploy applications across multiple regions for redundancy.

Popular Cloud Service Models

Service Model	Description	Example
IaaS (Infrastructure as a Service)	Provides virtual machines, networking, and storage.	AWS EC2, Azure Virtual Machines
PaaS (Platform as a Service)	Provides a platform to develop, run, and manage applications.	AWS Elastic Beanstalk, Azure App Services
SaaS (Software as a Service)	Cloud-based software solutions.	Google Drive, Microsoft Office 365

Example: Checking System Information on a Cloud-Based UNIX Server

uname -a

df -h

free -m

Exercise

1. List **three cloud-based UNIX services** available on AWS and Azure.
2. Use uname -a to check the operating system on a cloud-based UNIX instance.

Case Study: Migrating from On-Premise to Cloud Infrastructure

A financial firm moved **from physical servers to AWS EC2 instances running UNIX**. This shift reduced **hardware maintenance costs by 50%** while improving scalability and security.

CHAPTER 2: DEPLOYING UNIX VIRTUAL MACHINES ON AWS AND AZURE

1. Creating a UNIX Virtual Machine in AWS (EC2 Instance)

AWS provides **Elastic Compute Cloud (EC2)** for deploying UNIX-based virtual machines.

A. Steps to Launch an AWS EC2 Instance

1. **Login to AWS Console** – Go to AWS Management Console.
2. **Navigate to EC2** – Click on **"EC2" > "Launch Instance"**.
3. **Choose an Amazon Machine Image (AMI)** – Select a UNIX-based OS such as:
    - Ubuntu Server
    - Amazon Linux
    - Red Hat Enterprise Linux (RHEL)
4. **Select an Instance Type** – Choose a configuration (e.g., t2.micro for free-tier).
5. **Configure Network and Security** – Open SSH (port 22) for remote access.
6. **Add Storage and Tags** – Define disk space and add labels for identification.
7. **Launch and Connect** – Use SSH to connect:
8. ssh -i my-key.pem ubuntu@ec2-public-ip

2. Creating a UNIX Virtual Machine in Azure

Microsoft Azure provides **Virtual Machines (VMs)** to deploy UNIX servers.

A. Steps to Launch an Azure Virtual Machine

1. **Login to Azure Portal** – Go to Azure Portal.
2. **Navigate to Virtual Machines** – Click on **"Create a virtual machine"**.
3. **Choose an Image** – Select a UNIX OS such as:
    - Ubuntu 22.04
    - CentOS 8
    - SUSE Linux
4. **Set Instance Size** – Select the VM type (B1ls for free-tier).
5. **Configure Networking** – Allow SSH access (port 22).
6. **Review and Launch** – Click on **"Create",** then access the VM using SSH:
7. ssh azureuser@vm-public-ip

Exercise

1. Launch a **UNIX-based virtual machine** in AWS or Azure.
2. Connect to the instance using ssh.

Case Study: Deploying UNIX Servers for Web Hosting

A web hosting provider **migrated its services to AWS EC2 UNIX instances,** reducing deployment time from **days to minutes** and improving site performance.

Chapter 3: Configuring Networking and Security in Cloud UNIX Servers

1. Configuring Security Groups and Firewalls

Security is critical when working on cloud-based UNIX systems.

A. Configuring AWS Security Groups

To allow only SSH and HTTP traffic:

aws ec2 authorize-security-group-ingress --group-id sg-12345 --protocol tcp --port 22 --cidr 0.0.0.0/0

aws ec2 authorize-security-group-ingress --group-id sg-12345 --protocol tcp --port 80 --cidr 0.0.0.0/0

B. Configuring Azure Network Security Groups

Allow SSH access using the Azure CLI:

az network nsg rule create --resource-group myResourceGroup --nsg-name myNetworkSecurityGroup --name allow-ssh --priority 100 --protocol Tcp --direction Inbound --destination-port-range 22

2. Setting Up Firewalls on UNIX Instances

A. Using UFW on Ubuntu

sudo ufw allow OpenSSH

sudo ufw enable

B. Using Firewalld on RHEL/CentOS

sudo firewall-cmd --add-service=http --permanent

sudo firewall-cmd --reload

Exercise

1. Configure **security groups** in AWS or Azure to allow SSH access.
2. Set up a **firewall** on your cloud-based UNIX instance.

Case Study: Enhancing Security for a UNIX Cloud Server

An online retailer faced **brute-force SSH attacks** on its cloud-based UNIX servers. By implementing **firewalls and security groups,** they blocked **unauthorized access,** reducing attack attempts by **80%**.

CHAPTER 4: DEPLOYING APPLICATIONS AND STORAGE SOLUTIONS IN THE CLOUD

1. Deploying a Web Application on a Cloud-Based UNIX Server

To install and run a simple Nginx web server on AWS or Azure UNIX instances:

```
sudo apt update

sudo apt install nginx -y

sudo systemctl start nginx
```

Check the running server:

```
curl http://localhost
```

2. Using Cloud Storage with UNIX

A. Mounting an AWS S3 Bucket to a UNIX Server

```
sudo apt install s3fs

echo "ACCESS_KEY:SECRET_KEY" > ~/.passwd-s3fs

chmod 600 ~/.passwd-s3fs
```

s3fs my-bucket /mnt/s3 -o passwd_file=~/.passwd-s3fs

B. Mounting an Azure Blob Storage to a UNIX Server

sudo apt install blobfuse

mkdir ~/azureblob

blobfuse ~/azureblob --container-name=mycontainer --tmp-path=/mnt/resource/blobfusetmp

Exercise

1. Deploy a **web application** on a cloud-based UNIX instance.
2. Configure **cloud storage** and mount an AWS S3 bucket.

Case Study: Running a Scalable Web Application in the Cloud

A SaaS company deployed its **web application on AWS UNIX instances** and used **S3 for storage,** reducing infrastructure costs and improving scalability.

CONCLUSION

This guide covered:

✅ **Deploying UNIX-based virtual machines on AWS and Azure.**
✅ **Configuring networking and security for cloud instances.**
✅ **Deploying applications and managing cloud storage solutions.**
✅ **Using automation tools to optimize cloud operations.**

HANDS-ON CAPSTONE PROJECT: DEPLOYING AND MANAGING A SCALABLE WEB APPLICATION ON A UNIX CLOUD SERVER

CHAPTER 1: INTRODUCTION TO THE CAPSTONE PROJECT

Objective of the Project

This capstone project is designed to provide **hands-on experience** in **deploying, managing, and optimizing a scalable web application** on a **UNIX-based cloud server** (AWS or Azure). By completing this project, you will gain expertise in **cloud computing, UNIX system administration, security, automation, and performance optimization**.

What You Will Learn

✅ Deploy a **UNIX-based virtual machine (EC2 in AWS or Azure VM)**.
✅ Set up a **web server (Nginx or Apache) and host a sample web application**.
✅ Configure **security, networking, and cloud storage solutions**.
✅ Automate deployment using **shell scripting and cloud automation tools**.
✅ Optimize server performance and monitor logs for troubleshooting.

Project Scenario

You have been hired as a **Cloud DevOps Engineer** for a startup launching a web application. Your task is to deploy a **secure, scalable, and optimized UNIX-based cloud infrastructure**. The

project will involve setting up **a cloud server, web server, database, firewall rules, automated backups, and monitoring tools**.

CHAPTER 2: SETTING UP A UNIX CLOUD SERVER ON AWS OR AZURE

1. Create a UNIX Virtual Machine

A. On AWS (EC2 Instance)

1. Login to AWS Console → Navigate to **EC2**.
2. Click **Launch Instance**.
3. Choose an **Amazon Machine Image (AMI)**:
    - **Ubuntu 22.04 LTS**
    - **Amazon Linux 2**
    - **Red Hat Enterprise Linux**
4. Select an **instance type** (t2.micro for free-tier).
5. Configure networking and **allow SSH (port 22)**.
6. Download the SSH key (.pem file) and launch the instance.
7. Connect to the instance:
8. ssh -i my-key.pem ubuntu@ec2-public-ip

B. On Azure (Virtual Machine)

1. Login to **Azure Portal** → Go to **Virtual Machines**.
2. Click **Create a Virtual Machine**.
3. Select **Ubuntu 22.04** as the OS.

4. Configure **size, networking, and security groups**.
5. Open SSH **port 22** for remote access.
6. Launch and connect via SSH:
7. ssh azureuser@vm-public-ip

Exercise

1. Deploy a UNIX cloud server on **AWS EC2 or Azure Virtual Machine**.
2. Connect to your instance using SSH.

Case Study: Deploying a Secure Cloud Infrastructure

A fintech startup needed **a secure, scalable UNIX cloud infrastructure** for its web application. By setting up **AWS EC2 instances with automated security policies**, they **reduced security risks and improved performance**.

CHAPTER 3: DEPLOYING A WEB SERVER AND DATABASE

1. Install and Configure a Web Server

A. Installing Nginx on Ubuntu

sudo apt update

sudo apt install -y nginx

sudo systemctl start nginx

sudo systemctl enable nginx

B. Installing Apache on CentOS

sudo yum install -y httpd

sudo systemctl start httpd

sudo systemctl enable httpd

C. Verify Web Server is Running

curl http://localhost

2. Deploy a Sample Web Application

1. Create a sample HTML page:
2. echo "<h1>Welcome to My Cloud Web Application</h1>" | sudo tee /var/www/html/index.html
3. Restart the web server:
4. sudo systemctl restart nginx # For Nginx
5. sudo systemctl restart httpd # For Apache
6. Open a browser and visit:
7. http://your-public-ip

3. Set Up a MySQL Database

A. Install MySQL on Ubuntu

sudo apt install -y mysql-server

sudo systemctl start mysql

sudo systemctl enable mysql

B. Secure MySQL Installation

sudo mysql_secure_installation

C. Create a Sample Database and User

mysql -u root -p

CREATE DATABASE webapp;

CREATE USER 'webuser'@'%' IDENTIFIED BY 'securepassword';

GRANT ALL PRIVILEGES ON webapp.* TO 'webuser'@'%';

FLUSH PRIVILEGES;

EXIT;

Exercise

1. Install **Nginx or Apache** and verify access.
2. Deploy a **sample web application** and configure MySQL.

Case Study: Hosting a Scalable Web App in the Cloud

A SaaS startup deployed its **web app using Nginx on AWS EC2** with a **MySQL database**. Using **load balancing and auto-scaling,** they handled **high traffic efficiently**.

CHAPTER 4: CONFIGURING SECURITY AND NETWORKING

1. Configure Firewall and Security Rules

A. On AWS Security Groups

aws ec2 authorize-security-group-ingress --group-id sg-12345 --protocol tcp --port 80 --cidr 0.0.0.0/0

aws ec2 authorize-security-group-ingress --group-id sg-12345 --protocol tcp --port 22 --cidr 0.0.0.0/0

B. Using UFW Firewall on Ubuntu

sudo ufw allow OpenSSH

sudo ufw allow 80/tcp

sudo ufw enable

2. Enable HTTPS with Let's Encrypt SSL

sudo apt install certbot python3-certbot-nginx -y

sudo certbot --nginx -d yourdomain.com

Exercise

1. Configure **firewall rules** for SSH and web traffic.
2. Secure the website with **SSL using Let's Encrypt**.

Case Study: Enhancing Cloud Security for a Government Website

A government agency implemented **firewalls and SSL encryption on UNIX cloud servers**, reducing **cyberattack risks by 80%**.

CHAPTER 5: AUTOMATING BACKUPS AND MONITORING LOGS

1. Automating Backups with Cron Jobs

Schedule automatic backups using tar:

crontab -e

Add the following entry for **daily backups**:

0 2 * * * tar -czf /backup/webapp_backup_$(date +\%F).tar.gz /var/www/html

2. Monitoring Logs for Troubleshooting

A. View Web Server Logs

sudo tail -f /var/log/nginx/access.log

B. Check System Logs

sudo journalctl -xe

Exercise

1. Set up an **automatic daily backup** using cron.
2. Monitor server logs using journalctl.

Case Study: Disaster Recovery in a Cloud Environment

A university hosting an **online learning platform** implemented **automated backups** and **log monitoring**, reducing downtime during failures.

CONCLUSION

This project covered:

✅ **Deploying a UNIX cloud server on AWS/Azure.**
✅ **Setting up a secure web application and database.**
✅ **Configuring security, networking, and SSL encryption.**
✅ **Automating backups and monitoring logs for troubleshooting.**

Assignment Solution: Design and Implement a UNIX-Based Server Solution for a Business

Objective

This assignment provides a **step-by-step guide** to **designing and implementing a UNIX-based server solution** for a business. The solution will cover **server deployment, security configuration, web hosting, database setup, automation, and monitoring** to ensure high performance, reliability, and security.

Step 1: Define Business Requirements

Before setting up a UNIX-based server, it's essential to **understand the business needs**.

1. Identify Key Requirements

- **Type of Business** (E-commerce, SaaS, Banking, etc.)
- **Number of Users/Clients** (How many customers/employees will access the server?)
- **Services Required** (Web hosting, database, file sharing, email, etc.)
- **Security Needs** (SSL, firewalls, user authentication)
- **Scalability and Backup Strategy**

2. Choose the UNIX Distribution

Business Type	Recommended UNIX Distribution

Web Hosting	**Ubuntu Server, CentOS, Debian**
Enterprise Apps	**Red Hat Enterprise Linux (RHEL)**
Cloud-Based	**Amazon Linux, Ubuntu Server**
Security-Focused	**FreeBSD, OpenBSD**

Step 2: Deploy a UNIX Server (On-Premises or Cloud)

1. Setting Up a Cloud-Based UNIX Server (AWS/Azure)

For businesses that prefer a **scalable and cost-effective** solution, a cloud-based UNIX server is ideal.

A. Launch a UNIX-Based Cloud Server in AWS (EC2 Instance)

1. **Login to AWS Console** → Navigate to **EC2**.
2. **Click on "Launch Instance"**.
3. **Choose an Amazon Machine Image (AMI)**:
    - Ubuntu Server 22.04
    - Amazon Linux 2
    - Red Hat Enterprise Linux
4. **Select Instance Type** (t2.micro for free-tier).
5. **Configure Security Groups** (Allow SSH and HTTP).
6. **Launch and Connect Using SSH**:
7. ssh -i my-key.pem ubuntu@ec2-public-ip

B. Deploy a UNIX Server on Azure (Virtual Machine)

1. **Login to Azure Portal** → Create a **Virtual Machine**.
2. **Choose OS**: Ubuntu 22.04, CentOS, or RHEL.
3. **Set Up Networking** (Allow SSH, HTTP/S traffic).
4. **Launch and Connect Using SSH**:
5. ssh azureuser@vm-public-ip

2. Setting Up an On-Premises UNIX Server

For businesses that need full **control and privacy**, an on-premises UNIX server is an option.

1. Download **Ubuntu Server** or **CentOS ISO**.
2. Install on a **physical machine** with at least:
    - **8GB RAM**
    - **500GB SSD storage**
    - **Quad-core CPU**
3. Configure **network settings, users, and permissions**.

Exercise

1. Deploy a UNIX-based server **on AWS or Azure** and connect using SSH.
2. Set up **a static IP address** for better accessibility.

Step 3: Install and Configure a Web Server

A web server is essential for **hosting websites, applications, or APIs**.

1. Install Apache or Nginx Web Server

A. Installing Apache on Ubuntu

sudo apt update

sudo apt install -y apache2

sudo systemctl start apache2

sudo systemctl enable apache2

B. Installing Nginx on CentOS

sudo yum install -y nginx

sudo systemctl start nginx

sudo systemctl enable nginx

2. Deploy a Sample Web Application

1. Create a web directory:
2. sudo mkdir -p /var/www/html/mybusiness
3. Create an index page:
4. echo "<h1>Welcome to My Business</h1>" | sudo tee /var/www/html/mybusiness/index.html
5. Restart the web server:
6. sudo systemctl restart apache2 # For Apache
7. sudo systemctl restart nginx # For Nginx

3. Configure Virtual Hosts for Multi-Site Hosting

Edit Apache configuration file:

sudo nano /etc/apache2/sites-available/mybusiness.conf

Add the following:

<VirtualHost *:80>

ServerAdmin admin@mybusiness.com

DocumentRoot /var/www/html/mybusiness

ServerName mybusiness.com

</VirtualHost>

Enable the configuration and restart Apache:

sudo a2ensite mybusiness.conf

sudo systemctl reload apache2

Exercise

1. Install **Apache or Nginx** and deploy a test webpage.
2. Configure a **virtual host** for a custom domain.

STEP 4: SETTING UP A DATABASE SERVER

1. Installing MySQL on Ubuntu

sudo apt install -y mysql-server

sudo systemctl start mysql

sudo systemctl enable mysql

2. Secure MySQL Installation

sudo mysql_secure_installation

3. Create a Sample Database and User

```
mysql -u root -p

CREATE DATABASE businessdb;

CREATE USER 'businessuser'@'%' IDENTIFIED BY 'securepassword';

GRANT ALL PRIVILEGES ON businessdb.* TO 'businessuser'@'%';

FLUSH PRIVILEGES;

EXIT;
```

Exercise

1. Install **MySQL** and create a **sample database**.
2. Create a **restricted user account** for database access.

STEP 5: IMPLEMENTING SECURITY MEASURES

1. Setting Up a Firewall

A. On Ubuntu (UFW Firewall)

```
sudo ufw allow OpenSSH

sudo ufw allow 80/tcp

sudo ufw allow 443/tcp

sudo ufw enable
```

B. On CentOS (Firewalld)

```
sudo firewall-cmd --add-service=http --permanent

sudo firewall-cmd --add-service=https --permanent
```

sudo firewall-cmd --reload

2. Enabling SSL with Let's Encrypt

sudo apt install certbot python3-certbot-apache -y

sudo certbot --apache -d mybusiness.com

Exercise

1. Configure **firewall rules** for SSH, HTTP, and HTTPS.
2. Enable **SSL certificates** for secure web access.

STEP 6: AUTOMATING BACKUPS AND MONITORING

1. Automating Backups Using Cron Jobs

Edit crontab:

crontab -e

Add a daily backup job:

0 2 * * * tar -czf /backup/mybusiness_backup_$(date +\%F).tar.gz /var/www/html

2. Monitoring Logs and System Performance

- View system logs:
- sudo journalctl -xe
- Monitor real-time server performance:
- top
- htop # If installed

Exercise

1. Set up an **automated daily backup** using cron.
2. Monitor **web server logs** and troubleshoot errors.

CONCLUSION

This guide covered:

✅ **Deploying a UNIX-based server in the cloud or on-premises.**
✅ **Setting up a web server (Apache/Nginx) and hosting a business website.**
✅ **Configuring a MySQL database for business applications.**
✅ **Implementing security measures (firewalls, SSL encryption).**
✅ **Automating backups and monitoring server performance.**

Assignment Solution: Develop a Fully Automated Shell Script for System Maintenance

Objective

This assignment provides a **step-by-step guide** to creating a fully automated **shell script** for **system maintenance** on a UNIX/Linux system. The script will perform essential tasks such as **disk cleanup, log rotation, system updates, backup management, and monitoring resource usage**.

Step 1: Define the Requirements of the Maintenance Script

A system maintenance script should include the following features:

- ✅ **Cleaning up temporary files and logs** to free up disk space.
- ✅ **Rotating logs** to prevent excessive storage usage.
- ✅ **Updating the system** (package updates and security patches).
- ✅ **Monitoring system resource usage** (CPU, memory, and disk).
- ✅ **Performing automated backups** of critical files.
- ✅ **Sending reports via email** to system administrators.

Step 2: Writing the Shell Script

1. Create the Script File

Create a new script file using the following command:

nano system_maintenance.sh

2. Add the Shebang Line and Set Execution Permissions

Start the script with the shebang (#!/bin/bash), which tells the system to use Bash to execute the script.

#!/bin/bash

Make the script executable:

chmod +x system_maintenance.sh

Step 3: Implementing Each Maintenance Task

1. Define Variables

Define necessary directories and log files:

LOG_FILE="/var/log/system_maintenance.log"

BACKUP_DIR="/backup"

MAX_BACKUPS=5

EMAIL="admin@example.com"

2. Clean Temporary Files and Logs

To remove unnecessary system files:

echo "Cleaning up temporary files..." | tee -a $LOG_FILE

sudo rm -rf /tmp/*

sudo rm -rf /var/tmp/*

To remove system logs older than **30 days**:

sudo find /var/log -type f -mtime +30 -exec rm -f {} \;

3. Rotate System Logs

To prevent log files from consuming excessive disk space:

```
echo "Rotating logs..." | tee -a $LOG_FILE

sudo logrotate -f /etc/logrotate.conf
```

4. Update the System

For **Debian/Ubuntu-based** systems:

```
echo "Updating system packages..." | tee -a $LOG_FILE

sudo apt update -y && sudo apt upgrade -y
```

For **RHEL/CentOS-based** systems:

```
sudo yum update -y
```

5. Monitor System Resource Usage

To log **CPU, memory, and disk usage**:

```
echo "Logging system resource usage..." | tee -a $LOG_FILE

echo "CPU Usage:" >> $LOG_FILE

top -b -n1 | grep "Cpu(s)" >> $LOG_FILE


echo "Memory Usage:" >> $LOG_FILE

free -m >> $LOG_FILE


echo "Disk Usage:" >> $LOG_FILE

df -h >> $LOG_FILE
```

6. Perform Automated Backups

To back up critical system files:

```
echo "Performing system backup..." | tee -a $LOG_FILE

tar -czf $BACKUP_DIR/system_backup_$(date +%F).tar.gz /etc /home /var/www
```

To limit the number of backups and delete older ones:

```
ls -t $BACKUP_DIR/system_backup_*.tar.gz | tail -n +$((MAX_BACKUPS+1)) | xargs rm -f
```

7. Send Maintenance Report via Email

To send a maintenance report:

```
echo "Sending maintenance report via email..." | tee -a $LOG_FILE

mail -s "System Maintenance Report" $EMAIL < $LOG_FILE
```

Step 4: Testing the Script

After writing the script, **test it manually**:

```
./system_maintenance.sh
```

Verify that:

- ✅ Temporary files and logs are deleted.
- ✅ Logs are rotated.
- ✅ System updates are performed.
- ✅ Resource usage is logged.
- ✅ Backup is created successfully.
- ✅ Email report is sent.

Step 5: Automating the Script with Cron Jobs

To automate the script and run it **daily at midnight,** add it to the cron scheduler:

crontab -e

Add the following line:

o o * * * /path/to/system_maintenance.sh

To verify scheduled cron jobs:

crontab -l

Final Shell Script: Fully Automated System Maintenance

```
#!/bin/bash


# System Maintenance Script

LOG_FILE="/var/log/system_maintenance.log"

BACKUP_DIR="/backup"

MAX_BACKUPS=5

EMAIL="admin@example.com"


echo "Starting system maintenance on $(date)" | tee -a $LOG_FILE


# Clean Temporary Files and Logs
```

```
echo "Cleaning up temporary files..." | tee -a $LOG_FILE
sudo rm -rf /tmp/*
sudo rm -rf /var/tmp/*
sudo find /var/log -type f -mtime +30 -exec rm -f {} \;

# Rotate Logs
echo "Rotating logs..." | tee -a $LOG_FILE
sudo logrotate -f /etc/logrotate.conf

# Update the System
echo "Updating system packages..." | tee -a $LOG_FILE
sudo apt update -y && sudo apt upgrade -y

# Monitor System Resource Usage
echo "Logging system resource usage..." | tee -a $LOG_FILE
echo "CPU Usage:" >> $LOG_FILE
top -b -n1 | grep "Cpu(s)" >> $LOG_FILE

echo "Memory Usage:" >> $LOG_FILE
free -m >> $LOG_FILE
```

```
echo "Disk Usage:" >> $LOG_FILE

df -h >> $LOG_FILE


# Perform Backups

echo "Performing system backup..." | tee -a $LOG_FILE

tar -czf $BACKUP_DIR/system_backup_$(date +%F).tar.gz /etc
/home /var/www


# Remove Old Backups

ls -t $BACKUP_DIR/system_backup_*.tar.gz | tail -n
+$((MAX_BACKUPS+1)) | xargs rm -f


# Send Email Report

echo "Sending maintenance report via email..." | tee -a $LOG_FILE

mail -s "System Maintenance Report" $EMAIL < $LOG_FILE


echo "System maintenance completed successfully on $(date)" | tee
-a $LOG_FILE
```

CONCLUSION

This assignment covered:

✅ **Writing a fully automated shell script for system maintenance.**

✅ **Implementing disk cleanup, log rotation, system updates, monitoring, and backups.**

✅ **Testing the script manually and automating it with cron jobs.**

✅ **Sending maintenance reports via email.**