**Cloud Computing Lab Manual(BCSE1207)**

**Practical 1(A): Introduction to Cisco Packet Tracer**

**Scenario**: A small office is setting up its internal network to allow basic communication and file sharing between employees. As a network technician, your task is to create a small LAN setup and verify end-to-end connectivity.

**Use Cases**:  
 Learning IP addressing and connectivity.  
 Troubleshooting basic connection issues.  
 Understanding switch-based LAN communication.

**Application**: Simulate a simple LAN environment to understand device connectivity.

**Steps**:

* + Install Cisco Packet Tracer.
  + Add two PCs and a switch from the device list.
  + Connect devices using Copper Straight-Through cables.
  + Assign static IPs (e.g., 192.168.1.1 and 192.168.1.2).
  + Use the command prompt on PCs to ping each other.
  + Observe successful ICMP replies.

**Objective**: Understand basic LAN architecture, device connectivity, and IP configuration.

**Outcome**: Students will be able to construct and troubleshoot a basic network in Packet Tracer.

**Assignment**: Design a LAN with 5 PCs, 1 switch, assign IPs, verify connectivity using ping, and document your setup with a screenshot and explanation.

**Practical 1(B): Understanding Network Topologies and PDU Flow**

**Scenario**: A medium-sized business is evaluating different network topologies to optimize communication between its departments. As part of the IT team, you are required to simulate different network topologies (Star, Mesh, Bus) using Packet Tracer. You will use Simple and Complex PDUs to analyze how data travels across the network, evaluating packet delivery time, redundancy, and communication reliability.

**Use Cases**:

* + Designing campus or enterprise networks.
  + Analysing network performance before implementation.
  + Learning redundancy and fault tolerance.

**Application**: Explore how network design impacts communication efficiency and reliability using Packet Tracer simulation.

**Steps**:

* 1. Create and simulate a Star topology using a central switch and multiple end devices. Test communication using a Simple PDU.
  2. Create a Mesh topology with interconnected devices. Use a Complex PDU to evaluate routing paths and packet delivery.
  3. Simulate a Bus topology to observe single-path communication flow.
  4. Record and compare observations for packet delivery speed, redundancy, and communication reliability across topologies.

**Objective**: To explore various network topologies and understand the use of Simple and Complex PDUs in network communication analysis.

**Outcome**: Students will gain an understanding of different network topologies, learn how to simulate them, and analyze communication using Simple and Complex PDUs.

**Assignment**: Create and test each topology. Compare and document performance metrics (packet delivery time, redundancy, communication speed) in a table format using screenshots and a brief analysis.  
  
 **Practical 2:Router-Based Network Integration with DHCP & DNS Configuration**

**Scenario:** The company’s branches—Head Office, Branch A, and Branch B—need to be connected via routers. You are tasked with creating the network topology using routers, configuring DHCP for dynamic IP address allocation, and setting up a DNS server for name resolution between the branches.

**Steps:**

1. Use Packet Tracer to simulate three networks, each connected by routers.
2. Configure DHCP on each network to dynamically assign IP addresses.
3. Set up a DNS server to resolve hostnames to IP addresses.
4. Test inter-network communication to ensure connectivity and DNS functionality.

**Objective:** To demonstrate how to connect multiple networks using routers and configure DHCP and DNS for dynamic IP allocation and domain name resolution.

**Outcome:** Students will understand how to connect multiple networks using routers, configure DHCP for dynamic IP assignment, and set up DNS for hostname resolution.  
  
**Assignment:** Design three separate LANs connected via routers in Packet Tracer. Configure DHCP on each network and set up a central DNS server. Test inter-network communication and hostname resolution between PCs across networks. Document all IP settings, DHCP leases, DNS resolution logs, and include screenshots of working pings and DNS queries.

**Practical 3: Virtualization using VMware (Installing Ubuntu 22.04)**

**Scenario**: A software development team needs to test applications in a Linux environment on their Windows machines without altering their main OS. As a team member, you are asked to install and configure Ubuntu inside a virtual environment.

**Use Cases**:

* + OS-level sandboxing.
  + Safe experimentation and testing.
  + Training and simulation environments.

**Application**: Explore virtualization through OS-level abstraction.

**Steps**:

* 1. Install VMware Workstation/Player.
  2. Download Ubuntu 22.04 ISO.
  3. Create a new virtual machine and allocate 2GB RAM.
  4. Mount ISO and install Ubuntu.
  5. Configure basic settings post-install.

**Objective**: Get hands-on with virtualization concepts.

**Outcome**: Ability to install and manage VMs.

**Assignment**: Install Ubuntu and create a new user with specific permissions.

**Practical 4: Basic Linux Commands & Shell Scripting**

**Scenario**: A system administrator needs to automate log file generation and backup processes on a Linux server. You are tasked with writing a basic shell script to perform repetitive file management tasks.

**Use Cases**:

* + Command-line navigation and automation.
  + File system and permission management.
  + Cron job preparation.

**Application**: Use CLI and shell scripting to manage and automate tasks.

**Steps**:

* 1. Launch Ubuntu VM.
  2. Practice basic commands: ls, pwd, cd, touch, nano, chmod, rm.
  3. Create a shell script to create and archive multiple files.
  4. Run the script and verify output.

**Objective**: Build foundational Linux skills.

**Outcome**: Write basic scripts for task automation.

**Assignment**: Script to take user input to create 3 folders and generate log file.

**Practical 5: Shared Folder, Snapshot, and Backup in VirtualBox**

**Real-World Scenario**: A developer frequently modifies project files in a VM and needs a way to back up work and recover from errors. You are required to use shared folders and snapshots to secure development progress.

**Use Cases**:

* + File sharing between host and guest OS.
  + Version control and recovery.
  + Environment rollback and testing.

**Application**: Use VirtualBox features to share files and manage system states.

**Steps**:

* 1. Enable shared folder from host to guest.
  2. Access it from Linux VM using mount or auto-mount feature.
  3. Take snapshot before installing new software.
  4. Revert to snapshot if errors occur.

**Objective**: Learn backup/recovery and folder syncing.

**Outcome**: Manage VM state effectively.

**Assignment**: Create a file in shared folder and demonstrate recovery using snapshot.

**Practical 6: Docker and Basic Commands**

**Scenario**: A DevOps team wants to run isolated environments for testing code without spinning up full virtual machines. You are responsible for setting up Docker and exploring its core functionality.

**Use Cases**:

* + Lightweight application testing.
  + Consistent development environments.
  + Simplified CI/CD pipelines.

**Application**: Use Docker containers for isolated environments.

**Steps**:

* 1. Install Docker.
  2. Pull an image: docker pull ubuntu.
  3. Run container: docker run -it ubuntu bash.
  4. List containers/images.

**Objective**: Learn containerization basics.

**Outcome**: Run and manage containers.

**Assignment**: Pull and run Nginx container and access its default web page.

**Practical 7: Deploy Web App Using Docker**

**Scenario**: A startup wants to deploy their company website in a fast and portable manner using containerized solutions. You are in charge of containerizing a simple HTML-based site using Docker.

**Use Cases**:

* + Web deployment using Docker.
  + Dockerfile creation and port mapping.
  + App delivery without dependency conflicts.

**Application**: Container-based web app deployment.

**Steps**:

* 1. Write a simple HTML app.
  2. Create a Dockerfile with base image nginx.
  3. Copy HTML to container and expose port.
  4. Build and run image.

**Objective**: To teach students how to create a containerized environment for web application deployment using Docker. This includes writing a basic HTML page, building a Docker image with a Nginx server, and deploying the container to serve the web page.

**Outcome**: Students will gain practical skills in using Docker for packaging and deploying web applications. They will understand how to write Dockerfiles, expose application ports, and manage containers for lightweight deployment.

**Assignment**: Create a simple HTML website (one page with text and styling), write a Dockerfile to serve it using the Nginx image, build the Docker image, and run it exposing port 80. Access the site from a browser using localhost. Submit the Dockerfile, build and run commands, and screenshots showing the webpage in the browser.

**Practical 8: S3 and Static Website Hosting**

**Scenario**: A freelancer wants to host a personal portfolio website on a reliable and cost-effective platform. You are required to upload your portfolio to AWS S3 and configure it for static hosting.

**Use Cases**:

* + Cost-effective web hosting.
  + Learning object storage and bucket policies.
  + Serverless deployment.

**Application**: Use AWS S3 to deploy static websites.

**Steps**:

* 1. Create a bucket in AWS S3.
  2. Upload HTML/CSS files.
  3. Enable static hosting and make public.
  4. Access via the S3 website endpoint.

**Objective**: To introduce students to cloud-based object storage and demonstrate how AWS S3 can be used to host static websites. The activity focuses on creating buckets, uploading web files, setting permissions, and enabling static hosting.

**Outcome**: Students will be able to deploy static websites using AWS S3, configure bucket policies, and manage access control for public-facing web content. They will also understand the difference between object storage and server-based hosting.

**Assignment**: Design a basic portfolio website using HTML and CSS. Upload all files to an AWS S3 bucket. Enable static website hosting and configure the bucket policy for public access. Submit the S3 website URL and screenshots of the AWS S3 console showing bucket settings and file structure.

**Practical 9: EC2 and Static Website Deployment**

**Scenario**: A local business wants full control over its website's backend by hosting it on a virtual server. You are asked to configure an EC2 instance, install a web server, and host a static site.

**Use Cases**:

* + Learning cloud compute setup.
  + Installing and managing web servers.
  + Understanding security groups and SSH.

**Application**: Launch an EC2 instance and serve content.

**Steps**:

* 1. Launch Ubuntu EC2 instance.
  2. SSH into the instance.
  3. Install Apache/Nginx.
  4. Copy HTML files and start web service.

**Objective**: To provide hands-on experience with provisioning virtual servers on AWS using EC2, installing a web server (Apache or Nginx), and serving a static website from an instance. The practical emphasizes remote access via SSH and security configuration.

**Outcome**: Students will learn how to launch and manage EC2 instances, connect to them via SSH, install required software, and host static content. They will also understand the role of security groups and the public IP for accessing cloud-hosted services.

**Assignment**: Launch an EC2 instance using Ubuntu. Connect to the instance via SSH and install either Apache or Nginx. Upload a static HTML website to /var/www/html (or appropriate directory). Start the server and verify that the site is accessible via the public IP. Submit connection steps, commands used, and the site URL with screenshots.

**Practical 10: Deploy a MERN Application**

**Real-World Scenario**: A development team needs to deploy a full-stack project for client review and feedback. You are responsible for provisioning the infrastructure and deploying a MERN stack.

**Use Cases**:

* + Full-stack app deployment on cloud.
  + Hands-on with frontend/backend integration.
  + MongoDB hosting (local or cloud).

**Application**: Deploy a cloud-based MERN app.

**Steps**:

* 1. Set up EC2 or Docker environment.
  2. Install Node.js, MongoDB (or Atlas), and required dependencies.
  3. Clone a MERN app from GitHub.
  4. Start backend and frontend.

**Objective**: To enable students to deploy a full-stack web application using the MERN stack (MongoDB, Express, React, Node.js). This includes setting up infrastructure (Docker or EC2), installing dependencies, running frontend and backend servers, and connecting to a database.

**Outcome**: Students will understand the structure of full-stack applications, the relationship between frontend and backend services, and how to deploy them in the cloud. They will also learn how to connect to MongoDB, either locally or via a service like MongoDB Atlas.

**Assignment**: Deploy a pre-built MERN application (e.g., a blog or task manager). Set up the environment using either EC2 or Docker. Ensure that the frontend and backend are both running and accessible (on different ports or subpaths). Connect to a MongoDB database, run the app, and test the full workflow. Submit deployment steps, GitHub repo link, environment setup, and live demo URLs.