**----------------What is a Computer?**

A computer is an electronic device that processes data according to a set of instructions called programs. It can perform a wide range of tasks, from simple calculations to complex simulations, by executing instructions provided by software applications.

**CPU (Central Processing Unit)**:

referred to as the brain of the computer, carries out instructions from programs by performing basic arithmetic, logical, control, and input/output operations.  **Control Unit (CU)**: Directs operations of the processor by telling other parts of the computer how to respond to the instructions received.

 **Arithmetic Logic Unit (ALU)**: Performs arithmetic and logical operations.

**RAM (Random Access Memory)**:  **Volatility**: Data is lost when the computer is turned off.

 **Speed**: Faster than permanent storage like hard disks or SSDs.

**Hard Disk**:

* **Function**: A hard disk is a non-volatile storage device that stores data permanently.
* **Types**:
  + **HDD (Hard Disk Drive)**: Uses spinning magnetic disks to read/write data.
  + **SSD (Solid State Drive)**: Uses flash memory, offering faster data access
* **SMPS (Switched-Mode Power Supply)**: converts electrical power from an outlet into a usable form for the computer's internal components.  Ensuring stable power supply to the motherboard, CPU, and peripherals.

 **Motherboard**:

* **Function**: The motherboard is the main circuit board that connects all components of the computer, allowing communication between the CPU, RAM, storage, and peripheral devices.

**Cabinet**:

* **Function**: The cabinet, or case, is the enclosure that houses and protects all the internal components of the computer.

**Monitor**:

* **Function**: The monitor is the display device that allows users to view the output from the computer.
* **Types**:
  + **Technologies**: LCD, LED, OLED.

**Keyboard and Mouse**:

* **Function**: Input devices that allow users to interact with the computer.
* **Purpose**: Navigate and interact with graphical elements on the screen.

**Charles Babbage (1837)**:

* Often considered the "father of the computer,"

**Uses of a Computer:**

1. **Store Data**: Computers can store vast amounts of data, including documents, images, videos, and applications, on various storage devices like hard drives and SSDs.
2. **Access Data**: Computers provide quick and efficient access to stored data, allowing users to retrieve and use information as needed.
3. **Transfer Data**: Computers enable the transfer of data between devices and over networks, facilitating communication and data sharing.

### How Computers Came to Be

#### The Need for Storing Information

**Mankind was in need to store “Information.”**

As human societies evolved, there was a growing need to preserve knowledge, experiences, and data to ensure continuity, learning, and progress.

#### What is Information?

**Information includes:**

* **Experiences**: Personal and collective knowledge gained through events and activities.
* **Data**: Facts, figures, and observations collected from the environment.

#### Why Store Information?

**To pass it to other people or their next generations:**

* **Knowledge Transfer**: Sharing valuable insights, skills, and traditions.
* **Historical Records**: Preserving history for future reference.
* **Educational Purposes**: Teaching and learning from past experiences.

#### Evolution of Information Storage Methods

**Languages > Writings > Inscriptions & Maple Leaves > Paper!**

1. **Languages**:
   * **Verbal Communication**: The earliest form of sharing information through spoken words.
   * **Oral Tradition**: Stories, lessons, and knowledge passed down verbally from one generation to another.
2. **Writings**:
   * **Early Scripts**: Development of symbols and characters to represent sounds and words.
   * **Cuneiform and Hieroglyphs**: Examples of ancient writing systems used by Sumerians and Egyptians.
3. **Inscriptions & Maple Leaves**:
   * **Inscriptions**: Carving or engraving information on stone, metal, or other durable materials.
   * **Maple Leaves and Other Natural Materials**: Writing on leaves, bark, and other plant materials as a medium to record information.
4. **Paper**:
   * **Invention of Paper**: Credited to the Chinese, paper provided a more portable, durable, and versatile medium for writing.
   * **Books and Manuscripts**: Compilation of written works into bound volumes for easier dissemination and preservation.

**We are migrating from Paper to Computer age!**

This transition reflects the shift from traditional, physical means of storing information to digital, electronic formats.

#### Traditional Paper Storage

* **Paper**: Historically, information was recorded and stored on paper in the form of books, documents, and notes. While paper is tangible and easy to use, it has limitations in terms of storage capacity, durability, and ease of access.

#### Advantages of Digital Storage

* **Increased Capacity**: Digital media can store vast amounts of data in a compact form.
* **Durability**: Digital storage is less prone to physical damage compared to paper.
* **Portability**: Digital storage devices are easy to carry and transfer between systems.
* **Accessibility**: Information stored digitally can be quickly accessed, searched, and shared.

**Floppy Disk**:

* **Capacity**: Typically holds 1.44 MB of data

**CD/DVD Drive**:

* **Capacity**: CDs can hold up to 700 MB

**Pen Drive (USB Flash Drive)**:

* **Capacity**: Varies from a few MBs to several GBs storing personal files, documents, and media.

**Memory Cards**:

* **Capacity**: Ranges from a few GBs to several hundreds of GBs.
* **Usage**: Commonly used in cameras, smartphones, and other portable devices.

### How We Are Handling Information

#### Information Handling Methods

1. **Simplex Communication**:
   * **Example**: Radio & TV
   * **Description**: Communication is unidirectional. Information is transmitted from one side to the other, but there is no capability for the receiver to send information back.

**Half Duplex Communication**:

* **Example**: Walkie-Talkie
* **Description**: Communication can occur in both directions, but not simultaneously. At any given time, the communication channel can either send or receive information, but not both.

**Full Duplex Communication**:

* **Example**: Landline/Mobile/Internet Communication
* **Description**: Communication can occur in both directions simultaneously. Both parties can send and receive information at the same time.
* **Usage**: Telecommunication systems like landline phones, mobile phones, and internet communication, enabling real-time conversations and data exchange.

### What is Data?

**Data** is a collection of information. It can be numbers, text, images, audio, or any other form of information that can be processed by a computer. Data serves as both the input and the output of computer programs.

 **rogram**:

* A set of instructions written in a programming language.
* Defines a sequence of operations for the computer to perform.
* Example: A program to calculate the sum of two numbers.

 **Application Software**:

* A collection of programs designed to perform specific tasks.
* Divided into:
  + **System Software**: Manages the hardware and basic operations of a computer (e.g., Operating System).
  + **Application Software**: Performs specific user-oriented tasks (e.g.,

### Operating System (OS)

An **Operating System** is software that acts as an interface between the user and the computer hardware. It manages the computer's resources and provides a platform for running application software.

#### Types of Operating Systems:

* **Linux**:
  + An open-source operating system based on Unix.
  + Known for its stability, security, and flexibility.
  + Widely used for servers, desktops, and embedded systems.
* **Microsoft Windows**:
  + A widely used operating system for personal computers.
  + Known for its user-friendly interface and extensive software compatibility.
  + Includes versions like Windows 10, Windows 11, and Windows Server.
* **Apple macOS**:
  + The operating system used on Apple's Mac computers.
  + Known for its sleek design, strong integration with Apple hardware, and robust performance.
  + Includes versions like macOS Monterey and macOS Big Sur.

These operating systems provide the necessary environment for users to run applications and perform various tasks on their computers efficiently and securely.

### How Server Controls Client Computers

**Server computers** and **client computers** communicate and interact within a network. This relationship allows servers to control and manage client computers to perform various tasks.

#### Network Connection

1. **Network Setup**:
   * Servers and clients are connected through a network, which can be a local area network (LAN) or a wide area network (WAN).
   * Network components such as routers, switches, and network cables facilitate the connection.
2. **Server Configuration**:
   * The server computer is set up with specific software and configurations to manage and respond to client requests.
   * Servers often run specialized operating systems (e.g., Windows Server, Linux Server) that are designed to handle multiple client connections and provide various services.

#### Client-Server Interaction

1. **Configuration on Client Computers**:
   * Client computers are configured to connect to the server by specifying the server's network address and required authentication credentials.
   * Configuration settings can include IP addresses, DNS settings, and network protocols.
2. **Request-Response Model**:
   * **Client Requests**:
     + Client computers initiate requests to the server to access resources or services, such as files, applications, or databases.
     + Requests are typically made using network protocols like HTTP (web browsing), FTP (file transfer), or SMB (file sharing).
   * **Server Response**:
     + The server receives and processes the client requests.
     + Based on the request, the server provides the necessary resources, data, or services back to the client.
     + For example, a client requesting access to a file stored on the server will receive the file if the server approves the request and the client has the appropriate permissions.

#### Example Scenarios

1. **File Server**:
   * A server that stores and manages files.
   * Clients can upload, download, and share files with other clients through the server.
2. **Web Server**:
   * A server that hosts websites and web applications.
   * Clients (web browsers) request web pages from the server using the HTTP protocol, and the server responds by delivering the requested web pages.
3. **Database Server**:
   * A server that manages databases and handles database queries.
   * Clients (applications) send queries to the database server to retrieve, insert, update, or delete data. The server processes these queries and sends the results back to the clients.
4. **Email Server**:
   * A server that handles email sending, receiving, and storage.
   * Clients (email applications) connect to the email server to send and receive emails, manage mailboxes, and perform other email-related tasks.

#### Summary

In a client-server network, the server controls client computers by managing and responding to their requests for resources and services. This centralized control allows for efficient management of data, applications, and services, ensuring that clients can access the necessary resources as needed.

**Data Transfer Between Two Computers**

Data transfer between two computers involves several steps and components to ensure that information is correctly sent from one machine to another. Here’s a simplified overview of how this process works:

1. **Application Layer**:
   * The process starts at the application layer, where a program or service generates data to be sent.
   * Example: A web browser requests a web page, or an email client sends an email.
2. **Data Preparation**:
   * The data generated by the application is prepared for transmission.
   * This includes formatting the data into a specific protocol or format that can be understood by the receiving application.
3. **Network Interface Card (NIC)**:
   * The data is handed over to the NIC, which is responsible for handling the physical transmission of data over the network.
   * The NIC converts the data into packets that can be transmitted over the network.
4. **Packet Formation**:
   * Data is divided into smaller, manageable chunks called packets.
   * Each packet contains a portion of the data, as well as metadata like the source and destination addresses, sequence information, and error-checking data.
5. **Transfer in Cable**:
   * The packets are transferred over the network medium, such as an Ethernet cable, Wi-Fi, or fiber optics.
   * The physical layer of the network handles the actual transmission of the electrical or optical signals representing the packets.
6. **Receiving NIC**:
   * The receiving computer’s NIC captures the incoming packets from the network.
   * The NIC checks for errors and reassembles the packets into the original data.
7. **Data Handling**:
   * The reassembled data is passed up to the appropriate application or service on the receiving computer.
   * If any packets are missing or corrupted, the receiving NIC can request retransmission from the sender.
8. **Application Layer (Receiving)**:
   * The application layer receives the data and processes it according to its function.
   * Example: A web browser displays the requested web page, or an email client shows the received email.

### Data Transfer Between Two Computers

Data transfer between two computers involves several key concepts and components to ensure that information is correctly sent and received.

#### Network Devices

1. **Router**:

**The router used for connecting two or more networks**

* + Connects different networks and directs data packets between them.
  + Provides IP addressing and routing functions.

**Switch**:

1.multicast device which can send data to a particular pc you want

1. Connects devices within the same network and facilitates communication between them.
2. Operates at the data link layer (Layer 2) of the OSI model.

 **Modem**:

* Converts digital data from a computer into analog signals for transmission over phone lines or other communication media.
* Modulates and demodulates signals.

 **Network Interface Card (NIC)**:

* Hardware component that connects a computer to a network.
* Converts data into electrical signals for transmission over the network.

#### IP Addressing

1. **IPv4**:
   * 32-bit address divided into four octets (e.g., 192.168.1.1).
   * Supports around 4.3 billion addresses.
2. **IPv6**:
   * 128-bit address divided into eight groups of four hexadecimal digits (e.g., 2001:0db8:85a3:0000:0000:8a2e:0370:7334).
   * Supports a virtually unlimited number of addresses.
3. **Static IP**:
   * Assigned manually to a device and does not change.
4. **Dynamic IP**:
   * Assigned automatically by a DHCP server and may change periodically.

#### Subnetting

* **Subnetting**:
  + Divides a larger network into smaller, manageable sub-networks.
  + Improves network performance and security.
  + Utilizes subnet masks to determine the network and host portions of an IP address.
  + Example: 192.168.1.0/24 indicates a subnet mask of 255.255.255.0, where the first 24 bits represent the network portion.

#### Public Network

* **Public Network**:
  + Accessible by anyone on the internet.
  + Uses public IP addresses, which are globally unique and assigned by ISPs.
  + Example: Websites and online services use public IP addresses to be reachable from anywhere.

#### Private Network

* **Private Network**:
  + Restricted to a specific organization or group.
  + Uses private IP addresses, which are not routable on the public internet.
  + Example: 192.168.0.0/16, 172.16.0.0/12, and 10.0.0.0/8 are reserved for private networks.
  + Devices within a private network communicate with each other using private IP addresses.

#### Virtual Network

* **Virtual Network (VNet)**:
  + Created using virtualization technologies.
  + Allows multiple virtual machines (VMs) or containers to communicate as if they are on the same physical network.
  + Isolated from the physical network, providing enhanced security and flexibility.
  + Example: Virtual LAN (VLAN) segments a physical network into multiple logical networks.

**Virtual Private Network (VPN)**:

* Extends a private network across a public network, such as the internet.
* Encrypts data to ensure secure communication between remote users and the private network.
* Provides privacy and anonymity by masking the user's IP address.
* Example: Employees working remotely can securely access their company's internal resources using a VPN.

**Example Scenario of Data Transfer**

1. **Network Devices**:
   * Computer A and Computer B are connected via switches and routers.
2. **IP Addressing**:
   * Computer A has an IP address of 192.168.1.10, and Computer B has an IP address of 192.168.1.20.
3. **Subnetting**:
   * Both computers are in the same subnet, 192.168.1.0/24.
4. **Data Transfer**:
   * Computer A sends a data packet to Computer B.
   * The data packet is passed to Computer A's NIC, where it is encapsulated in an Ethernet frame.
   * The frame is sent to the switch, which forwards it to the router if necessary.
   * The router directs the packet to the appropriate destination based on the IP address.
   * Computer B's NIC receives the frame, extracts the packet, and passes the data to the appropriate application.

### Network Devices

#### RJ45

* **RJ45 Connector**:
  + An 8-pin connector commonly used to connect computers to Ethernet-based local area networks (LANs).
  + Plugs into an Ethernet port on a NIC or other network device.

**Ethernet Cable**:

* Used to physically connect network devices such as computers, switches, and routers.

**Network Interface Card (NIC)**:

* A hardware component that enables a computer or device to connect to a network.
* Converts data into electrical signals for transmission over the network and vice versa.
* Available in wired (Ethernet) and wireless (Wi-Fi) versions.

**Switch**:

* A network device that connects multiple devices within the same network and facilitates communication between them.
* Operates at the data link layer (Layer 2) of the OSI model.
* Uses MAC addresses to forward data to the correct destination.

**Router**:

* A network device that connects different networks and directs data packets between them.
* Operates at the network layer (Layer 3) of the OSI model.
* Uses IP addresses to route data to the correct destination.
* Often provides additional features such as DHCP, NAT, and firewall capabilities.

**Internet Modem**:

* Converts digital data from a computer into analog signals for transmission over phone lines or other communication media.
* Modulates and demodulates signals for internet connectivity.

**Gateway**:

* A network device that acts as a bridge between different networks, often combining the functionality of a modem and a router.
* Provides connectivity to the internet and routes traffic between internal networks and external networks.
* **Computer** is connected to the **Switch** via an **Ethernet Cable** using an **RJ45 Connector**.
* The **Switch** connects multiple devices within the same local network.
* The **Router** connects the local network to the internet via the **Internet Modem/Gateway**.
* The **NIC** in each device enables network communication.

Each device plays a crucial role in ensuring smooth and efficient data transfer within and beyond the local network.

1. **Step 1: Browser Request**: You type "[www.google.com](http://www.google.com)" into your web browser to request a web page.
2. **Step 2: DNS Resolution**: Your computer sends a request to a DNS (Domain Name System) server to translate "[www.google.com](http://www.google.com)" into an IP address. DNS servers maintain a directory of domain names and their corresponding IP addresses.
3. **Step 3: Obtaining Public IP**: The DNS server responds with the public IP address associated with "[www.google.com](http://www.google.com)". This IP address identifies the server hosting Google's website on the Internet.
4. **Step 4: Sending Packets**: Your computer's Network Interface Card (NIC) uses this public IP address to send packets (units of data) over your local network and then over the Internet to Google's server.
5. **Step 5: Receiving Response**: Google's server processes your request, retrieves the web page content, and sends it back to your computer's NIC in the form of packets.
6. **Step 6: Displaying Web Page**: Your web browser receives these packets, reconstructs the web page content, and displays it on your screen.

**What is an IP Address?**

An IP (Internet Protocol) address is a unique identifier assigned to each device connected to a network. It allows devices to communicate with each other over the internet or a local network.

**Types of IP Addresses**

There are two main versions of IP addresses:

1. **IPv4**: The most common type, consisting of four sets of numbers separated by periods (e.g., 192.168.1.1).
2. **IPv6**: A newer version designed to replace IPv4, consisting of eight groups of hexadecimal numbers separated by colons (e.g., 2001:0db8:85a3:0000:0000:8a2e:0370:7334).

We'll focus on IPv4 here for simplicity.

**Structure of an IPv4 Address**

An IPv4 address is a 32-bit number, typically represented in decimal format as four octets (8-bit numbers) separated by periods.

Example: 192.168.1.1

* **192** (first octet)
* **168** (second octet)
* **1** (third octet)
* **1** (fourth octet)

**Classes of IP Addresses**

IPv4 addresses are divided into five classes (A, B, C, D, E) based on their leading bits and intended usage. The most commonly used are Classes A, B, and C.

1. **Class A**:
   * Range: 1.0.0.0 to 126.255.255.255
   * Default Subnet Mask: 255.0.0.0
   * Used for large networks (many devices).
2. **Class B**:
   * Range: 128.0.0.0 to 191.255.255.255
   * Default Subnet Mask: 255.255.0.0
   * Used for medium-sized networks.
3. **Class C**:
   * Range: 192.0.0.0 to 223.255.255.255
   * Default Subnet Mask: 255.255.255.0
   * Used for small networks.

**Example Breakdown**

Let's take the IP address 192.168.1.1 and break it down:

* **Class**: It's a Class C address because it falls within the range 192.0.0.0 to 223.255.255.255.
* **Network Portion**: In a Class C address, the first three octets (192.168.1) represent the network part.
* **Host Portion**: The last octet (1) represents the host (specific device) within that network.

**Public IP Range:**

* Public IP addresses are assigned by Internet authorities and are routable on the Internet.
* They typically fall within these ranges:
  + Class A: 1.0.0.0 to 126.255.255.255
  + Class B: 128.0.0.0 to 191.255.255.255
  + Class C: 192.0.0.0 to 223.255.255.255

**Private IP Range:**

* Private IP addresses are used within private networks (like your home or office) and are not routable on the Internet.
* They fall within these reserved ranges specified by RFC 1918:
  + Class A: 10.0.0.0 to 10.255.255.255
  + Class B: 172.16.0.0 to 172.31.255.255
  + Class C: 192.168.0.0 to 192.168.255.255

These private IP ranges allow multiple devices to share a single public IP address (usually assigned by your ISP) through a router or gateway device, enabling local network communication without direct exposure to the Internet.

* **Definition**: An IP address is a numeric label assigned to each device (such as a computer, printer, or router) within a network that uses the Internet Protocol for communication.
* **Purpose**: It serves as a unique identifier, enabling devices to be recognized and located on a network or the Internet. IP addresses facilitate the routing of data packets between devices and networks.
* **Format**: IP addresses are typically written in dotted-decimal notation (e.g., 192.168.1.1 for IPv4 or 2001:0db8:85a3:0000:0000:8a2e:0370:7334 for IPv6). IPv4 addresses are 32-bit numbers, while IPv6 addresses are 128-bit numbers, allowing for a much larger number of unique addresses.
* **Functionality**: IP addresses enable devices to send and receive data packets across networks, ensuring that information reaches its intended destination based on the IP address's routing information.

Overall, IP addressing forms the backbone of modern network communication, providing the foundation for devices to connect, communicate, and exchange data reliably across the Internet and other interconnected networks.

Subnetting:

Subnetting is a technique used in computer networking to divide a larger network into smaller sub-networks called subnets. This process helps in efficiently managing IP addresses and network resources. Here's a basic example of subnetting:

1. **Steps for Subnetting with Additional 2 Bits**
2. **1. Determine Subnet Mask**
3. **Original Subnet Mask**: /24
4. **New Subnet Mask**: /24 + 2 bits = /26
5. **2. Choose Subnet Bit Allocation**
6. We have 8 bits available for the host portion in the /24 subnet mask.
7. We will use 2 of these 8 bits for subnetting, leaving 6 bits for host addresses.
8. **3. Calculate New Subnet Mask**
9. **New Subnet Mask**: /26
10. **Subnet Mask in Decimal**: 255.255.255.192
11. **4. Calculate Number of Subnets and Hosts per Subnet**
12. **Number of Subnets**: 22=42^2 = 422=4 subnets
13. **Hosts per Subnet**: 26−2=622^{6} - 2 = 6226−2=62 usable host addresses (subtracting network and broadcast addresses)
14. **5. Calculate Subnet Addresses**
15. Each subnet will have a block size of 26=642^{6} = 6426=64 addresses, including network and broadcast addresses.
16. Here are the subnets:
17. **Subnet Address Calculation**
18. **Subnet 1: 192.168.1.0/26**
19. Network Address: 192.168.1.0
20. First Usable IP: 192.168.1.1
21. Last Usable IP: 192.168.1.62
22. Broadcast Address: 192.168.1.63
23. **Subnet 2: 192.168.1.64/26**
24. Network Address: 192.168.1.64
25. First Usable IP: 192.168.1.65
26. Last Usable IP: 192.168.1.126
27. Broadcast Address: 192.168.1.127
28. **Subnet 3: 192.168.1.128/26**
29. Network Address: 192.168.1.128
30. First Usable IP: 192.168.1.129
31. Last Usable IP: 192.168.1.190
32. Broadcast Address: 192.168.1.191
33. **Subnet 4: 192.168.1.192/26**
34. Network Address: 192.168.1.192
35. First Usable IP: 192.168.1.193
36. Last Usable IP: 192.168.1.254
37. Broadcast Address: 192.168.1.255

LINUX:

The mkdir command in Linux is used to create new directories.

The id command in Linux is used to display the user ID (UID) and group ID (GID) of the current user or a specified user. It also shows the groups to which the user belongs.

**Common Options:**

* **-u**: Display only the effective user ID.
* **-g**: Display only the effective group ID.
* **-G**: Display all group IDs.
* **-n**: Display the name instead of the number (useful with -u, -g, or -G).
* **-r**: Display the real ID instead of the effective ID

**CentOS** (Community ENTerprise Operating System) is a popular, community-supported, free and open-source Linux distribution. It aims to provide a free, enterprise-class, community-supported computing platform functionally compatible with its upstream source, Red Hat Enterprise Linux (RHEL).

**What is Linux?**

Linux is an open-source operating system kernel, which serves as the foundation for a wide range of operating systems known as Linux distributions (or distros). It was originally created by Linus Torvalds in 1991 and has since grown to be one of the most widely used operating systems in the world.

**Key Features of Linux:**

1. **Open Source**:
   * The source code is freely available for anyone to view, modify, and distribute. This promotes collaboration and innovation within the community.
2. **Security**:
   * Linux has a robust security model with features such as user permissions, file permissions, and built-in firewalls.
3. **Stability and Reliability**:
   * Linux is known for its stability and can run for long periods without needing a reboot, making it ideal for servers and critical applications.
4. **Multitasking and Multiuser**:
   * Supports multiple users and can handle multiple tasks simultaneously without performance degradation.

 **ls** - List directory contents

* Example: ls -l
  + Lists files and directories in long format.

 **cd** - Change directory

* Example: cd /home/user
  + Changes the current directory to /home/user.

 **pwd** - Print working directory

* Example: pwd
  + Displays the current working directory.

 **mkdir** - Make directories

* Example: mkdir newdir
  + Creates a new directory named newdir.

 **rmdir** - Remove empty directories

* Example: rmdir emptydir
  + Removes the empty directory named emptydir.

 **touch** - Change file timestamps or create empty files

* Example: touch file.txt
  + Creates an empty file named file.txt.

 **rm** - Remove files or directories

* Example: rm file.txt
  + Deletes the file named file.txt.

 **cp** - Copy files or directories

* Example: cp file.txt newfile.txt
  + Copies file.txt to newfile.txt.

 **mv** - Move or rename files or directories

* Example: mv file.txt newlocation/
  + Moves file.txt to the directory newlocation/.

 **cat** - Concatenate and display file content

* Example: cat file.txt
  + Displays the contents of file.txt.

 **less** - View file content one page at a time

* Example: less largefile.txt
  + Views the content of largefile.txt one page at a time.

 **more** - View file content one screen at a time

* Example: more textfile.txt
  + Views the content of textfile.txt one screen at a time.

 **head** - Output the first part of files

* Example: head -n 10 file.txt
  + Displays the first 10 lines of file.txt.

 **tail** - Output the last part of files

* Example: tail -n 20 file.txt
  + Displays the last 20 lines of file.txt.

 **chmod** - Change file permissions

* Example: chmod 755 script.sh
  + Changes permissions of script.sh to read, write, execute for owner, and read, execute for others.

 **chown** - Change file owner and group

* Example: chown user:group file.txt
  + Changes the owner of file.txt to user and the group to group.

 **find** - Search for files in a directory hierarchy

* Example: find /home/user -name "\*.txt"
  + Searches for files with .txt extension under /home/user.

 **grep** - Search text using patterns

* Example: grep "pattern" file.txt
  + Searches for pattern in file.txt.

 **locate** - Find files by name

* Example: locate file.txt
  + Finds files named file.txt on the system.

 **man** - Display manual pages for commands

* Example: man ls
  + Displays the manual (documentation) for the ls command.

 **ps** - Report a snapshot of current processes

* Example: ps aux
  + Displays a list of all processes running on the system.

  **top** - Display Linux tasks

* Example: top
  + Displays real-time information about CPU, memory usage, and running processes.

 **kill** - Terminate a process

* Example: kill -9 PID
  + Terminates the process with ID PID.

 **df** - Report file system disk space usage

* Example: df -h
  + Displays disk space usage in a human-readable format.

 **du** - Estimate file space usage

* Example: du -sh directory
  + Shows disk usage (total size) of directory in a human-readable format.

 **free** - Display amount of free and used memory in the system

* Example: free -h
  + Displays memory usage in a human-readable format.

 **ifconfig** - Configure network interfaces

* Example: ifconfig eth0
  + Displays configuration details for the network interface eth0.

 **iwconfig** - Configure wireless network interfaces

* Example: iwconfig wlan0
  + Displays configuration details for the wireless network interface wlan0.

 **ping** - Send ICMP ECHO\_REQUEST to network hosts

* Example: ping google.com
  + Sends ICMP packets to google.com to check connectivity.

 **wget** - Download files from the web

* Example: wget https://example.com/file.zip
  + Downloads file.zip from https://example.com.

 **curl** - Transfer data from or to a server

* Example: curl -O https://example.com/file.txt
  + Downloads file.txt from https://example.com.

 **nano** - Simple text editor

* Example: nano newfile.txt
  + Opens newfile.txt in the nano text editor.

 **vi** or **vim** - Advanced text editor

* Example: vi file.txt
  + Opens file.txt in the vi text editor.

 **sudo** - Execute a command as another user, typically as the superuser

* Example: sudo apt-get update
  + Updates package lists using apt-get with superuser privileges.

 **apt-get** - Package handling utility for Debian-based distributions

* Example: sudo apt-get install package-name
  + Installs package-name using the apt package manager.

 **yum** - Package manager for RPM-based distributions

* Example: sudo yum install package-name
  + Installs package-name using the yum package manager.

 **tar** - Archive files

* Example: tar -czvf archive.tar.gz directory/
  + Creates a gzip-compressed tar archive of directory/.

 **zip** - Package and compress files

* Example: zip -r archive.zip directory/
  + Creates a zip archive of directory/ and its contents.

 **unzip** - Extract compressed files

* Example: unzip archive.zip
  + Extracts archive.zip into the current directory.

 **ssh** - OpenSSH SSH client

* Example: ssh user@hostname
  + Connects to hostname using SSH as user.

 **scp** - Secure copy (remote file copy program)

* Example: scp file.txt user@hostname:/remote/directory/
  + Copies file.txt to /remote/directory/ on hostname.

 **rsync** - Remote file and directory synchronization

* Example: rsync -avz source/ destination/
  + Syncs files and directories from source/ to destination/ recursively.

 **echo** - Display a line of text

* Example: echo "Hello, World!"
  + Prints Hello, World! to the terminal.

 **date** - Display or set the system date and time

* Example: date
  + Displays the current date and time.

 **cal** - Display a calendar

* Example: cal 07 2024
  + Displays the calendar for July 2024.

 **whoami** - Print the current user name

* Example: whoami
  + Prints the username of the current user.

 **hostname** - Show or set the system's host name

* Example: hostname
  + Displays the hostname of the system.

 **uname** - Print system information

* Example: uname -a
  + Displays detailed system information including kernel version.

 **history** - Show command history

* Example: history
  + Displays a list of previously executed commands.

 **Modes in Vi**:

* **Command Mode**: Default mode for navigating and entering commands.
* **Insert Mode**: Used for inserting or editing text within the file.

 **Navigating in Command Mode**:

* Use arrow keys or h, j, k, l for left, down, up, and right movements respectively.
* gg - Move to the beginning of the file.
* G - Move to the end of the file.
* :n - Move to line number n.

 **Editing in Insert Mode**:

* Press i to enter Insert Mode and start typing.
* Press Esc to return to Command Mode after editing.

 **Saving and Quitting**:

* To save changes and quit Vi:

:wq

To quit without saving:

:q!

**Vim (Vi Improved):**

Vim is an enhanced version of Vi with additional features and improvements.

1. **Opening a File**:
   * To open example.txt in Vim

 **Modes in Vim**:

* Similar to Vi, Vim has Command Mode and Insert Mode.
* In Command Mode, use navigation keys or Vim-specific commands.

 **Navigating in Command Mode**:

* Same as Vi: h, j, k, l, gg, G, :n.

 **Editing in Insert Mode**:

* Press i to enter Insert Mode and start typing.
* Additional modes include a (append after cursor), A (append at end of line), and others for specific editing tasks.

 **Saving and Quitting**:

* To save changes and quit Vim

:wq

To quit without saving:

:q!

Adit in one year

Whish storage class they appled

100mb/10 rupes in one month🡪standared storage class

Inteligentteir,standard in..arched storage class,glacer deep architecture

Automation using one storage class to another storage class

Transition policy changing

Versioning used for no data loss.with same name taking in files

S3 versioning:

Sqs simple que service

Phone lo storage full aythe pendrives taking and save data like google drive.untill 50 gb it will store the files.

Cloud aws there is no limit.based on data it give charges.it will gives 3 copies you restoring

Object by default prite.so create a public.

**What is IAM?**

IAM is a service in AWS that helps you manage who can access your AWS resources and what they can do with them.

**Key Components of IAM:**

1. **Users:**
   * **What They Are:** Individual people or applications that need access to AWS.
   * **Example:** An employee at your company who needs to manage AWS resources.
2. **Groups:**
   * **What They Are:** Collections of users.
   * **Purpose:** Makes it easier to manage permissions for multiple users at once.
   * **Example:** A group called "Developers" might include all the people who work on software development.
3. **Roles:**
   * **What They Are:** Special types of access that can be assigned to users, services, or applications.
   * **Purpose:** Allows temporary or special access without using personal credentials.
   * **Example:** An EC2 instance (a virtual server) might assume a role to access files in S3 (a storage service).
4. **Policies:**
5. An IAM Policy is a document that defines permissions. It specifies what actions are allowed or denied for specific AWS resources.

6.**Purpose:** Policies are used to control what a user, group, or role can do with AWS resources.

* + **What They Are:** Rules that define what actions are allowed or denied.
  + **Purpose:** Control what users or roles can do with AWS resources.
  + **Example:** A policy might allow a user to read data from a database but not write to it.

 **Root Account:**

* **Full Access** to everything.
* **Sign In:** Email address and root password.
* **Use:** Limited to account management and billing.

 **IAM User:**

* **Customizable Access** based on permissions.
* **Sign In:** Username and password.
* **Use:** Daily tasks and operations, with specific permissions for their role.

**1. Create a Group and Assign AWS Resource Permissions**

**Step 1: Create a Group**

1. **Log in to AWS Management Console:** Use your root account or an IAM user with sufficient permissions.
2. **Navigate to IAM:** Go to the IAM dashboard by searching for "IAM" in the AWS Management Console.
3. **Create a New Group:**
   * In the left-hand menu, select **"Groups"**.
   * Click **"Create New Group"**.
   * Enter a **Group Name** (e.g., MyDevelopersGroup).
   * Click **"Next Step"**.

**Step 2: Attach Permissions Policy**

1. **Attach Policies:**
   * Select the permissions you want to assign to the group. For example, you might select AmazonS3ReadOnlyAccess to allow read-only access to S3 buckets.
   * Click **"Next Step"**.
   * Review and click **"Create Group"**.

**2. Create a New User and Assign This User to the Group**

**Step 1: Create a New User**

1. **Navigate to IAM Users:**
   * In the IAM dashboard, select **"Users"**.
   * Click **"Add User"**.
   * Enter a **User Name** (e.g., newuser).
   * Select **"Access type"** options:
     + **Programmatic Access:** For API, CLI, SDK.
     + **AWS Management Console Access:** For console login.
   * Click **"Next: Permissions"**.

**Step 2: Assign User to Group**

1. **Add User to Group:**
   * Select the group you created earlier (e.g., MyDevelopersGroup).
   * Click **"Next: Tags"** (optional).
   * Click **"Next: Review"**.
   * Review the details and click **"Create User"**.
   * Save the credentials (Access Key ID and Secret Access Key) if you enabled programmatic access.

**3. Try to Log In via New IAM User and Check Permissions**

**Step 1: Log In to AWS Management Console**

1. **Log In as New User:**
   * Go to the [AWS Management Console sign-in page](https://aws.amazon.com/console/).
   * Use the new user credentials (username and password) to log in.
   * Verify the permissions by attempting to access resources specified by the group's policies.

**Step 2: Check Permissions**

1. **Verify Access:**
   * Try accessing S3 buckets or any other resources allowed by the policies attached to the group.
   * If the permissions are correct, you should be able to perform the actions specified in the policy.

**4. Understand Programmatic Access and IAM Account Settings**

**Programmatic Access:**

1. **Definition:** Programmatic access is for using AWS services through the API, CLI, or SDKs. It requires Access Key ID and Secret Access Key.
2. **Use Cases:** Automated scripts, applications, or tools that interact with AWS services.

**IAM Account Settings:**

1. **Password Policy:** Configure rules for user passwords, such as complexity and expiration.
2. **Multi-Factor Authentication (MFA):** Add an extra layer of security by requiring a second form of authentication.
3. **Account Recovery:** Settings for recovering access if credentials are lost.

**5. Install and Configure AWS CLI Using New User Account**

**Step 1: Install AWS CLI**

1. **Download and Install AWS CLI:**
   * Follow the instructions for your operating system from the [AWS CLI installation guide](https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html).

**Step 2: Configure AWS CLI**

1. **Open Command Line Interface:**
   * Run the aws configure command.
2. **Enter New User Credentials:**
   * **AWS Access Key ID:** Enter the Access Key ID for the new IAM user.
   * **AWS Secret Access Key:** Enter the Secret Access Key for the new IAM user.
   * **Default region name:** Enter your preferred region (e.g., us-east-1).
   * **Default output format:** Choose the output format (e.g., json).

aws configure

aws s3 ls

**. What is S3?**

Amazon S3 (Simple Storage Service) is a scalable object storage service offered by Amazon Web Services (AWS). It allows users to store and retrieve any amount of data, anytime, from anywhere on the web. S3 is designed for high availability and durability, making it ideal for a variety of use cases such as backups, data archiving, and content distribution.

**1. Scalability**

**S3**:

* **Automatic Scaling**: S3 automatically scales to handle any amount of data without manual intervention. You can store from a few megabytes to exabytes of data.
* **Elastic Capacity**: The storage capacity increases or decreases seamlessly based on your needs.

**Pendrives**:

* **Fixed Capacity**: USB flash drives have a fixed storage capacity (e.g., 32GB, 64GB, 128GB). You need to purchase additional drives if you require more space.
* **Manual Management**: You must manually manage storage upgrades by physically acquiring new drives.

**2. Durability and Availability**

**S3**:

* **Durability**: Provides 99.999999999% (11 9’s) durability over a year, meaning that data is extremely unlikely to be lost.
* **Availability**: Offers 99.99% availability of objects, ensuring data is accessible when needed.

**Pendrives**:

* **Physical Durability**: Susceptible to physical damage (e.g., breaking, data corruption) which can lead to data loss.
* **Availability**: Data is only available when you physically have the pendrive with you. If lost or damaged, accessing the data becomes impossible.

**3. Security**

**S3**:

* **Encryption**: Supports encryption both in-transit (using SSL/TLS) and at-rest (using server-side encryption).
* **Access Controls**: Provides robust access control features through IAM, bucket policies, and ACLs to manage who can access or modify your data.

**Pendrives**:

* **Physical Security**: Security depends on physical protection. Data can be easily accessed if the drive is lost or stolen.
* **Encryption**: Requires separate software or hardware encryption solutions to secure data, which is not inherently provided by the drive itself.

**4. Cost-Effective**

**S3**:

* **Pay-as-You-Go**: Charges are based on the amount of data stored, requests made, and data transferred. No upfront costs or long-term contracts.
* **Cost Efficiency**: You only pay for what you use, with the ability to scale up or down based on your needs.

**Pendrives**:

* **Upfront Costs**: Requires upfront purchase of drives. Costs can add up with larger capacities or multiple drives.
* **No Ongoing Fees**: No ongoing costs once purchased, but lacks the flexibility and scalability of cloud solutions.

**5. Flexibility**

**S3**:

* **Wide Range of Data Types**: Supports various data types and sizes, including large files, backup data, and media content.
* **Integration**: Easily integrates with other AWS services (e.g., data analysis, processing, machine learning) to enhance functionality and workflows.

**Pendrives**:

* **Limited Data Types**: Suitable for personal storage and transfer of files but lacks advanced integration options.
* **Standalone**: Not designed to integrate with other services or applications; primarily used for local data storage and transfer.

**1. Create a Bucket**

**Purpose**: Buckets are containers for storing objects (files) in Amazon S3. Each bucket must have a unique name globally.

**Steps**:

1. **Log in to AWS Management Console**: Go to the [AWS Management Console](https://aws.amazon.com/console/) and sign in.
2. **Navigate to S3**: In the console, search for and select "S3" to open the S3 management console.
3. **Create Bucket**:
   * Click the **"Create bucket"** button.
   * **Bucket Name**: Enter a unique name for your bucket (e.g., my-unique-bucket-name).
   * **Region**: Choose the AWS region where you want your bucket to be located.
   * **Configure Options**: You can skip most options or configure them as needed (e.g., versioning, logging).
   * **Set Permissions**: By default, buckets are private. You can adjust permissions later.
   * Click **"Create bucket"**.

**Example**: You create a bucket named my-website-bucket in the us-west-2 region.

**2. Upload a File (Object) and Check Properties**

**Purpose**: Store files in your bucket and view their metadata.

**Steps**:

1. **Open Your Bucket**: In the S3 management console, click on the bucket you created (my-website-bucket).
2. **Upload File**:
   * Click the **"Upload"** button.
   * **Add Files**: Click **"Add files"** and select a file from your computer (e.g., index.html).
   * **Review**: You can add tags or set permissions if needed.
   * Click **"Upload"** to start uploading the file.
3. **Check Properties**:
   * After upload, select the file (index.html) from the bucket.
   * Click on the **"Properties"** tab to view details such as size, last modified date, and storage class.

**Example**: You upload index.html and check its properties, noting that it's 2KB in size and was last modified today.

**3. Enable Public Bucket Access**

**Purpose**: Allow public access to the objects in your bucket.

**Steps**:

1. **Open Bucket Permissions**:
   * Select the bucket (my-website-bucket).
   * Go to the **"Permissions"** tab.
2. **Edit Bucket Policy**:
   * Click on **"Bucket Policy"** and add a policy to allow public access. Example policy:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Principal": "\*",

"Action": "s3:GetObject",

"Resource": "arn:aws:s3:::my-website-bucket/\*"

}

]

}

* + Click **"Save"**.

1. **Edit Block Public Access Settings**:
   * Click **"Block public access"** settings and ensure public access is allowed by unchecking the "Block all public access" option.
   * Click **"Save changes"**.

**Example**: After configuring, the objects in my-website-bucket are publicly accessible.

**4. Accessing All Objects in the Bucket Publicly**

**Purpose**: Retrieve and view files from your bucket over the internet.

**Steps**:

1. **Find Object URL**:
   * Select an object (e.g., index.html) from your bucket.
   * Click **"Copy URL"** or find the URL in the **"Object URL"** section in the Properties tab.
2. **Access Object**:
   * Paste the URL into your browser or use a tool like curl to access the file. For example: https://my-website-bucket.s3.amazonaws.com/index.html.

**Example**: Accessing https://my-website-bucket.s3.amazonaws.com/index.html will display the content of index.html.

**5. Deleting the Objects**

**Purpose**: Remove files from your bucket when they are no longer needed.

**Steps**:

1. **Select Object**:
   * Open your bucket (my-website-bucket).
   * Select the object you want to delete (e.g., index.html).
2. **Delete Object**:
   * Click **"Actions"** and then **"Delete"**.
   * Confirm the deletion by clicking **"Delete object"** in the confirmation dialog.

**Example**: You delete index.html from my-website-bucket, and it will no longer be accessible via its URL.

**How to create a Static website in S3?**

* **Create a sample HTML file.**
* **Create a bucket with Public access.**
* **Upload the sample HTML file to the bucket.**
* **Create a public access over the file (object ACL).**
* **Create a public access over the bucket.**

**Create a Sample HTML File**

**Purpose**: Create a simple HTML file to serve as your static website.

**. Create a Bucket with Public Access**

**Purpose**: Create an S3 bucket to store and serve your static website files.

**Steps**:

1. **Log in to AWS Management Console**: Go to the [AWS Management Console](https://aws.amazon.com/console/) and sign in.
2. **Navigate to S3**: Search for and select **"S3"**.
3. **Create a Bucket**:
   * Click the **"Create bucket"** button.
   * **Bucket Name**: Enter a unique name (e.g., my-static-website-bucket).
   * **Region**: Choose an AWS region.
   * **Bucket Settings**: You can skip configuring advanced options for now.
   * **Permissions**: Uncheck **"Block all public access"** to allow public access. Confirm that you want to allow public access by acknowledging the warning.
   * Click **"Create bucket"**.

**3. Upload the Sample HTML File to the Bucket**

**Purpose**: Upload your HTML file to the newly created bucket.

**Steps**:

1. **Open Your Bucket**: In the S3 console, click on the bucket you created (my-static-website-bucket).
2. **Upload File**:
   * Click the **"Upload"** button.
   * **Add Files**: Click **"Add files"** and select the index.html file you created.
   * **Review and Upload**: Click **"Upload"** to start the upload process.

**4. Create a Public Access over the File (Object ACL)**

**Purpose**: Make the uploaded HTML file publicly accessible.

**Steps**:

1. **Select the Object**: In your bucket, click on the index.html file.
2. **Edit Permissions**:
   * Go to the **"Permissions"** tab.
   * Under **"Public access"**, click **"Edit"**.
   * Check **"Public access"** to allow public access. If the object policy doesn't show, you might need to edit the bucket policy to allow public access.
   * Click **"Save changes"**.

**Example**: Set the index.html file to be publicly accessible so anyone with the URL can view it.

**5. Create Public Access over the Bucket**

**Purpose**: Ensure that the bucket allows public access to the objects stored in it.

**Steps**:

1. **Open Bucket Permissions**:
   * In your bucket, go to the **"Permissions"** tab.
2. **Block Public Access Settings**:
   * Click on **"Block public access"** settings.
   * Ensure that the **"Block all public access"** option is unchecked.
   * Confirm that you understand the risks and click **"Save changes"**.

**Example**: Allow public access to your bucket so that anyone can access the files stored within it, including the index.html file.

**Final Step**: To view your static website, you need to configure the bucket for static website hosting:

1. **Enable Static Website Hosting**:
   * In the S3 bucket properties, go to the **"Static website hosting"** section.
   * Click **"Enable"**, and specify index.html as the index document.
   * Click **"Save changes"**.

**Access URL**: Use the website endpoint provided by S3 to view your static site. For example, it might be http://my-static-website-bucket.s3-website-us-west-2.amazonaws.com.

**1. S3 Standard**

**Purpose**: Ideal for frequently accessed data with high durability and availability.

**Characteristics**:

* **Durability**: 99.999999999% (11 9’s) over a year.
* **Availability**: 99.99% of objects.
* **Access**: Low latency and high throughput.
* **Cost**: Higher cost compared to other classes due to its high performance.

**Example**: Storing website assets like images, videos, and frequently accessed database backups that are accessed regularly.

**Usage**: Suitable for applications that require high availability and performance.

**2. S3 Intelligent-Tiering**

**Purpose**: Automatically optimizes costs by moving data between two access tiers based on changing access patterns.

**Characteristics**:

* **Durability**: 99.999999999% (11 9’s) over a year.
* **Availability**: 99.9% for the frequent tier and 99% for the infrequent tier.
* **Access**: Moves data between frequent and infrequent access tiers automatically.
* **Cost**: Higher cost for frequent access, lower cost for infrequent access, with a small monitoring and automation fee.

**Example**: Storing data with unpredictable access patterns, like logs or data that might be accessed occasionally.

**Usage**: Ideal for data where access patterns are unpredictable or unknown.

**3. S3 Standard-IA (Infrequent Access)**

**Purpose**: Designed for data that is less frequently accessed but requires rapid access when needed.

**Characteristics**:

* **Durability**: 99.999999999% (11 9’s) over a year.
* **Availability**: 99.9% of objects.
* **Access**: Lower cost for storage but higher cost for retrieval.
* **Cost**: Lower cost than S3 Standard, with a retrieval fee.

**Example**: Storing backups or long-term data that you don’t access frequently but need to be available when required.

**Usage**: Suitable for backups and disaster recovery data where access is rare but retrieval speed is essential.

**4. S3 One Zone-IA**

**Purpose**: Lower-cost alternative for infrequent access data that doesn’t need to be stored in multiple availability zones.

**Characteristics**:

* **Durability**: 99.999999999% (11 9’s) over a year, but only within one availability zone.
* **Availability**: 99.5% of objects.
* **Access**: Lower cost for storage with retrieval fees, but data is not replicated across multiple zones.
* **Cost**: Cheaper than S3 Standard-IA, due to single availability zone storage.

**Example**: Storing secondary backups or data that can be easily recreated if lost, like development artifacts or temporary files.

**Usage**: Ideal for infrequently accessed data where lower costs are more important than resilience across multiple zones.

**5. S3 Glacier**

**Purpose**: Cost-effective storage for archival data that is rarely accessed, with retrieval times ranging from minutes to hours.

**Characteristics**:

* **Durability**: 99.999999999% (11 9’s) over a year.
* **Availability**: 99.99% of objects.
* **Access**: Designed for long-term archival storage with retrieval times of minutes to hours.
* **Cost**: Much lower cost compared to S3 Standard, with a fee for retrieval.

**Example**: Archiving old data, regulatory compliance records, or historical logs that are infrequently accessed but need to be stored for long periods.

**Usage**: Suitable for data that is rarely accessed but must be retained for compliance or historical purposes.

**6. S3 Glacier Deep Archive**

**Purpose**: Lowest-cost storage class for long-term archival of data that is rarely accessed.

**Characteristics**:

* **Durability**: 99.999999999% (11 9’s) over a year.
* **Availability**: 99.9% of objects.
* **Access**: Designed for long-term archival with retrieval times ranging from hours to days.
* **Cost**: Lowest cost for storage, with a retrieval fee and longer retrieval times compared to S3 Glacier.

**Example**: Storing data for compliance or legal reasons, such as tax records or historical documents, where access is extremely rare.

**Usage**: Ideal for data that is archived for the long term and where cost is the primary concern.

S3 Versioning is a powerful feature that allows you to keep multiple versions of an object in the same bucket, which helps in recovering from accidental deletions, overwrites, or application failures.

**Example Scenario: Managing Document Versions**

**Scenario**: Suppose you're managing a bucket where you store a file called report.pdf. You enable S3 Versioning to ensure that you can keep track of changes and recover previous versions if needed.

**1. Enable Versioning on Your Bucket**

1. **Open the S3 Console**: Go to the [AWS Management Console](https://aws.amazon.com/console/) and open the S3 service.
2. **Select Your Bucket**: Click on the bucket where you want to enable versioning (e.g., my-documents-bucket).
3. **Go to Properties**: Click on the **"Properties"** tab.
4. **Enable Versioning**:
   * Find the **"Bucket Versioning"** section.
   * Click **"Edit"** and then select **"Enable"**.
   * Click **"Save changes"**.

Now versioning is enabled for the bucket.

**2. Upload an Object**

1. **Upload report.pdf**:
   * Click on the **"Upload"** button in your bucket.
   * Choose report.pdf from your local system and click **"Upload"**.

At this point, this is the first version of the file.

**3. Modify and Re-upload the Object**

1. **Update the File**:
   * Make some changes to report.pdf (e.g., update content).
   * Save the updated file as report.pdf.
2. **Re-upload the Updated File**:
   * Go back to your bucket.
   * Click **"Upload"** again and upload the updated report.pdf.

With versioning enabled, S3 will now store this as a new version of report.pdf. The original file is still preserved, and both versions are stored in the bucket.

**4. Check the Versions**

1. **View Versions**:
   * Click on the **"Show versions"** button in your bucket view.
   * You will see multiple versions of report.pdf, each with a unique version ID.
2. **Example View**:
   * **Version ID: a1b2c3d4...** – Contains the content of the first upload.
   * **Version ID: e5f6g7h8...** – Contains the content of the updated upload.

**6. Delete an Object**

1. **Delete report.pdf**:
   * Select the current version of report.pdf in the bucket.
   * Click **"Actions"** and select **"Delete"**.

The object is marked as deleted, but the previous versions are still retained.

**7. View Deleted Object**

1. **View Deleted Object**:
   * Even after deletion, you can see previous versions by enabling **"Show versions"**.
   * Previous versions of report.pdf are still available and can be retrieved or restored.

**Benefits**: With S3 Versioning, you can recover from accidental deletions, overwrites, and other unintended changes by accessing previous versions of your objects.

Amazon S3 Replication allows you to automatically copy objects from one S3 bucket to another, either within the same AWS account or across different accounts. This feature is useful for data backup, disaster recovery, and compliance requirements.

UDC project:

PHP application that collects user data and uploads files to Amazon S3

**Navigate to the Web Server Directory**

1. **Open Terminal**: Access your web server via SSH or terminal.
2. **Change Directory**

cd /var/www/html

**Create composer.json File**

1. **Create and Edit File**

vi composer.json

1. **Add the Following**

{

"require": {

"aws/aws-sdk-php": "^3.0"

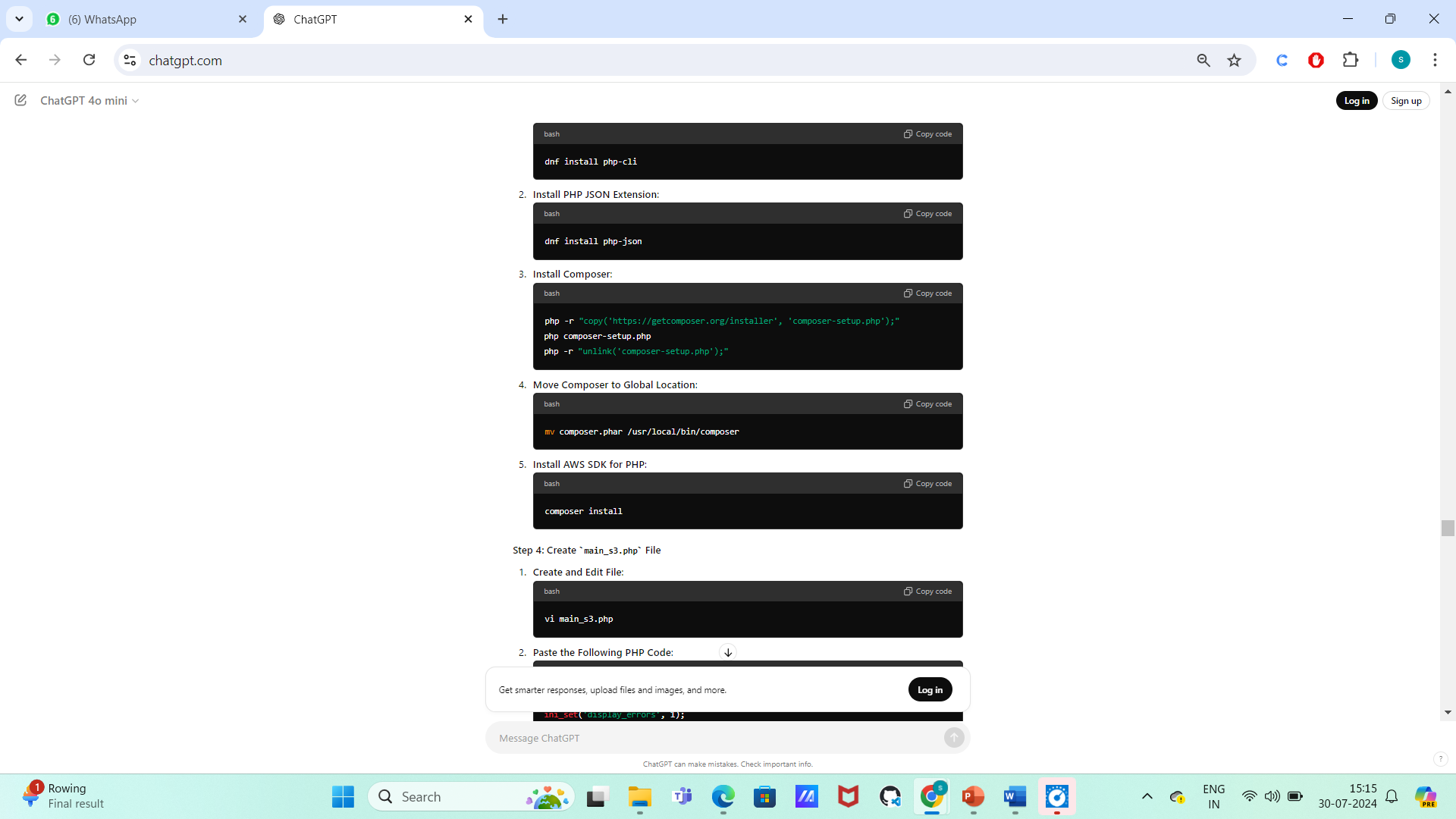
}

}

* + Press i to enter insert mode in vi.
  + Paste the above content.
  + Press Esc, then type :wq to save and exit.

**tep 3: Install Required Packages**

1. **Install PHP CLI**



* + Press i to enter insert mode in vi.
  + Paste the PHP code.
  + Press Esc, then type :wq to save and exit.

**Summary**

1. **Navigate to /var/www/html**: Set your working directory.
2. **Create composer.json**: Define dependencies for AWS SDK for PHP.
3. **Install Required Packages**: Set up PHP CLI, JSON, Composer, and install dependencies.
4. **Create main\_s3.php**: Implement the PHP script to handle user data collection and file upload to S3.

### ****Amazon EC2 – Introduction****

**Amazon Elastic Compute Cloud (EC2)** is a core service in AWS, providing scalable virtual computing resources in the cloud. It allows users to create and manage virtual machines, also known as instances, within AWS’s data centers.

**Key Features of EC2:**

1. **Virtual Machines**:
   * **Instances**: EC2 instances are virtual servers that run applications on the AWS cloud. They provide the compute capacity required for your workloads and can be customized in terms of CPU, memory, storage, and network performance.
2. **Data Centers**:
   * **Regions and Availability Zones (AZs)**: AWS data centers are organized into regions and availability zones. A region is a geographical area with multiple AZs, which are isolated locations within a region. EC2 allows you to deploy instances in various regions and AZs to ensure high availability and disaster recovery.
3. **Security and Resizability**:
   * **Security**: EC2 instances benefit from AWS’s security features, including Virtual Private Cloud (VPC) for network isolation, security groups for firewall rules, and IAM roles for access control.
   * **Resizability**: You can change the instance type or size as your needs change. This flexibility allows you to scale resources up or down based on your workload requirements.

**Different Types of EC2 Instances**

**Flexible Pricing Models**

**Explanation: EC2 offers several pricing models to help optimize costs based on usage patterns.**

**Real-Time Example:**

* On-Demand Instances: Pay for compute capacity by the hour or second with no long-term commitments. A developer uses On-Demand Instances to test a new application.
* Reserved Instances: Significant discount for a one- or three-year commitment. A SaaS provider uses Reserved Instances for steady-state workloads to save costs.
* Spot Instances: Purchase unused EC2 capacity at up to 90% discount. A data analytics company uses Spot Instances for processing large datasets that can be interrupted.
* Savings Plans: Flexible pricing model offering significant savings over On-Demand Instances for a consistent amount of usage. A cloud-native startup uses Savings Plans to reduce their compute costs.

**EC2 Instance Creation – LINUX OS**

**Step 1: Create an EC2 Instance**

1. **Log into AWS Management Console**
   * Go to the [AWS Management Console](https://aws.amazon.com/console/).
   * Log in with your AWS account credentials.
2. **Navigate to EC2 Dashboard**
   * In the Services menu, find and select **EC2** under the Compute section.
3. **Launch an Instance**
   * Click on the **Launch Instance** button.
4. **Choose an Amazon Machine Image (AMI)**
   * Select a Linux-based OS. For example, you can choose Amazon Linux 2 or Ubuntu.
5. **Select an Instance Type**
   * Choose **t2.micro** (which is free tier eligible).
   * Click **Next: Configure Instance Details**.

**Step 2: Configure Instance Details**

1. **Configure Instance Settings**
   * You can leave the default settings, or configure as needed (e.g., number of instances, network settings).
   * Click **Next: Add Storage**.

**Step 3: Add Storage**

1. **Modify Storage (if needed)**
   * The default settings are usually fine. Click **Next: Add Tags**.

**Step 4: Add Tags**

1. **Tag Your Instance (optional)**
   * You can add tags to help organize and identify your instances.
   * Click **Next: Configure Security Group**.

**Step 5: Configure Security Group**

1. **Create a Security Group**
   * Add a rule to allow inbound traffic for SSH (port 22) and HTTP (port 80). You can also add HTTPS (port 443) if you plan to use it later.
   * For SSH, set the source to your IP (for security).
   * Click **Review and Launch**.

**Step 6: Review and Launch**

1. **Review Configuration**
   * Ensure all settings are correct, then click **Launch**.

**Step 7: Create Key Pair**

1. **Create a Key Pair**
   * In the dialog that appears, select **Create a new key pair**.
   * Give it a name, and click **Download Key Pair** (this will download a .pem file).
   * Click **Launch Instances**.

**Step 8: Connect to the Instance**

1. **Download PuTTY and PuTTYgen**
   * Download PuTTY from [here](https://www.putty.org/).
   * Download PuTTYgen from [here](https://puttygen.com/download.php?val=49).
2. **Convert .pem to .ppk**
   * Open PuTTYgen.
   * Click **Load** and select the .pem file you downloaded earlier (change file type to All Files to see it).
   * Click **Save private key** (you can choose to save it with a .ppk extension).
3. **Open PuTTY**
   * In the **Hostname** field, enter ec2-user@<public-IP-of-your-EC2-instance>. You can find the public IP in the EC2 dashboard.
   * In the left pane, navigate to **Connection > SSH > Auth** and browse for the .ppk file you just saved.
4. **Connect to the Instance**
   * Click **Open** to connect. If a security alert pops up, click **Yes** to continue.

**Step 9: Gain Root Access**

1. **Switch to Root User**
   * After connecting, type sudo su to switch to the root user. This gives you administrative privileges.

**Step 10: Install Apache HTTP Server**

1. **Install Apache**
   * Type yum install httpd -y (for Amazon Linux) or apt-get install apache2 -y (for Ubuntu) to install the Apache web server.

**Step 11: Start Apache Service**

1. **Start Apache**
   * Type service httpd start (for Amazon Linux) or systemctl start apache2 (for Ubuntu) to start the web server.

**Ebs:**

**Amazon EBS (Elastic Block Store)**

* Integration: EBS volumes are designed to be used with Amazon EC2 instances. You can attach EBS volumes to EC2 instances, where they appear as traditional block devices.
* Persistence: Unlike the instance store volumes that come with some EC2 instances and are ephemeral (data is lost when the instance is terminated), EBS volumes are persistent. This means data remains on the EBS volume even if the associated EC2 instance is stopped or terminated.
* Snapshots: EBS supports creating snapshots of volumes, which are backups stored in Amazon S3. These snapshots can be used to restore volumes or create new volumes.
* Scalability: EBS volumes can be resized, and you can change the volume type (e.g., from standard HDD to SSD) based on your performance and cost requirements.

Block-level storage like Amazon EBS provides a robust, flexible, and high-performance way to manage storage for applications. By dividing data into blocks and managing them independently, it offers efficiency and high speed for a variety of use cases, especially in cloud environments where scalability and persistence are key**.**

HDD AND SSD:

* **Performance**: SSDs are faster in terms of read/write speeds and latency, while HDDs are slower due to mechanical parts.
* **Capacity**: HDDs generally offer larger storage capacities at a lower cost, but SSDs are catching up in both capacity and cost.
* **Durability**: SSDs are more durable due to the lack of moving parts, while HDDs are more susceptible to physical damage.
* **Power Consumption and Noise**: SSDs are more power-efficient and silent compared to HDDs.

Choosing between an HDD and an SSD typically depends on your needs for speed, capacity, budget, and the type of tasks you'll be performing. For tasks requiring high performance, such as gaming or video editing, SSDs are usually preferred, while HDDs are often used for bulk storage or in scenarios where cost per GB is a major consideration.

**Why Use EBS?**

1. **Create a File System**:
   * **What It Means**: EBS allows you to create a file system on a virtual disk. This means you can store files and directories just like you would on a traditional hard drive.
   * **Example**: If you need a place to store and organize files for a web server or application, you can set up an EBS volume, format it with a file system (like ext4 for Linux or NTFS for Windows), and use it to manage your files.
2. **Run an Operating System**:
   * **What It Means**: EBS volumes can be used to boot an EC2 instance. Essentially, they provide the disk space where the operating system (OS) and other necessary files are stored.
   * **Example**: When you launch an EC2 instance, it typically uses an EBS volume as the root volume where the OS is installed. This setup ensures that the OS and any system settings persist even if the EC2 instance is stopped or terminated.
3. **Run a Database**:
   * **What It Means**: EBS volumes can be used to store the data for databases. They provide the high-speed, reliable storage needed for database operations.
   * **Example**: For a high-performance application like an e-commerce site, you might use an EBS volume to store the database data, ensuring quick access and processing of transactions.
4. **Install Required Applications (Storing Data)**:
   * **What It Means**: You can install software and store data on EBS volumes. This is useful for any application that requires persistent storage for its data.
   * **Example**: If you are running a web application that needs to save user-uploaded files, or if you need to store logs or other application data, you would use EBS to keep this data safe and accessible.

**Advantages of Amazon EBS with Simple Examples**

1. **High Availability (HA)**
   * **Automatic Replication**: Think of EBS like a hard drive in a computer, but even more reliable. If the computer's hard drive has a backup system that automatically keeps a copy of your files in the same room, you wouldn't lose your files if the hard drive fails. Similarly, EBS keeps copies of your data within the same Availability Zone (AZ) so that if one part fails, your data is still accessible from another part of the same AZ.
   * **Example**: Imagine you’re running a website on an EC2 instance using an EBS volume. If there's a hardware issue in the AZ where your data is stored, EBS automatically keeps your data safe and accessible, so your website keeps running smoothly.
2. **Designed for Production Environments**
   * **Performance and Reliability**: EBS is like using a high-quality storage drive designed to handle constant use. It’s built to ensure that your data is always quickly accessible and reliable, which is essential for applications that can’t afford downtime.
   * **Example**: If you’re managing an online store with EBS, you need the storage to be fast and reliable so that customers don’t experience delays while browsing or making purchases. EBS delivers the consistent performance needed for such critical operations.
   * **Durability**: EBS is like having a very durable and secure safe for your most important documents. It’s engineered to keep your data safe over time, protecting it from loss.
   * **Example**: If you're using EBS to store sensitive financial records, you can trust that EBS will keep these records safe and intact, thanks to its high durability.
3. **Dynamically Scalable**
   * **Increase Capacity**: EBS allows you to easily grow your storage space as needed without interrupting your services. It’s like adding extra shelves to a storage room without having to stop using the room.
   * **Example**: If your application starts receiving more users and needs more storage, you can increase the size of your EBS volume to handle the extra data without having to shut down your server.
   * **Change Volume Type**: EBS lets you switch to different types of storage depending on what you need. It’s like being able to upgrade from regular shelves to high-performance ones that can hold heavier and more sensitive items.
   * **Example**: If you need faster access to your data for a new, high-performance application, you can change your EBS volume from a standard HDD to a high-speed SSD without any downtime.
   * **No Downtime**: EBS changes can be made without affecting your running applications. It’s like renovating a room while you’re still using the house – the work doesn’t disrupt your daily activities.
   * **Example**: If you need to resize or upgrade your EBS volume, you can do so without stopping your website or application, ensuring that your users are not impacted.
   * **Performance Consistency**: Adjustments to EBS volumes won’t hurt the performance of your applications. It’s like having a car that runs smoothly no matter how much you change or upgrade its parts.
   * **Example**: If you’re upgrading the storage for a database, EBS makes sure that the performance of your database remains steady and reliable throughout the upgrade process.

**IOPS (Input/Output Operations Per Second)**

**Definition**: IOPS measures how many read and write operations a storage system can perform per second. It's a metric of how many individual data operations can be completed in a given second.

**Importance**:

* **Performance Indicator**: Higher IOPS means the storage can handle more operations simultaneously, which is crucial for applications that require high-speed data access.
* **Use Cases**: Applications with high transaction rates, such as databases and high-performance computing tasks, benefit from high IOPS.

**Example**:

* **Scenario**: Imagine you have a database that needs to process thousands of transactions per second. A storage system with high IOPS will be able to handle these transactions efficiently without becoming a bottleneck.

**Throughput**

**Definition**: Throughput measures the amount of data that can be read from or written to the storage system per second. It is usually expressed in megabytes per second (MBps) or gigabytes per second (GBps).

**Importance**:

* **Data Transfer Rate**: Throughput indicates how much data can be transferred over a period, which is essential for tasks that involve large data sets or large files.
* **Use Cases**: Applications that handle large files or large-scale data transfers, such as data analytics and video processing, benefit from high throughput.

**Example**:

* **Scenario**: If you’re running a data analysis application that processes large datasets, high throughput will ensure that the data can be read from or written to the storage quickly, leading to faster processing times.

 **IOPS** measures how many read/write operations the storage can handle per second. High IOPS is crucial for applications with many small, random data operations.

1.  **Throughput** measures the volume of data transferred per second. High throughput is important for applications handling large files or large data transfers.

**BS SSD Volume Types**

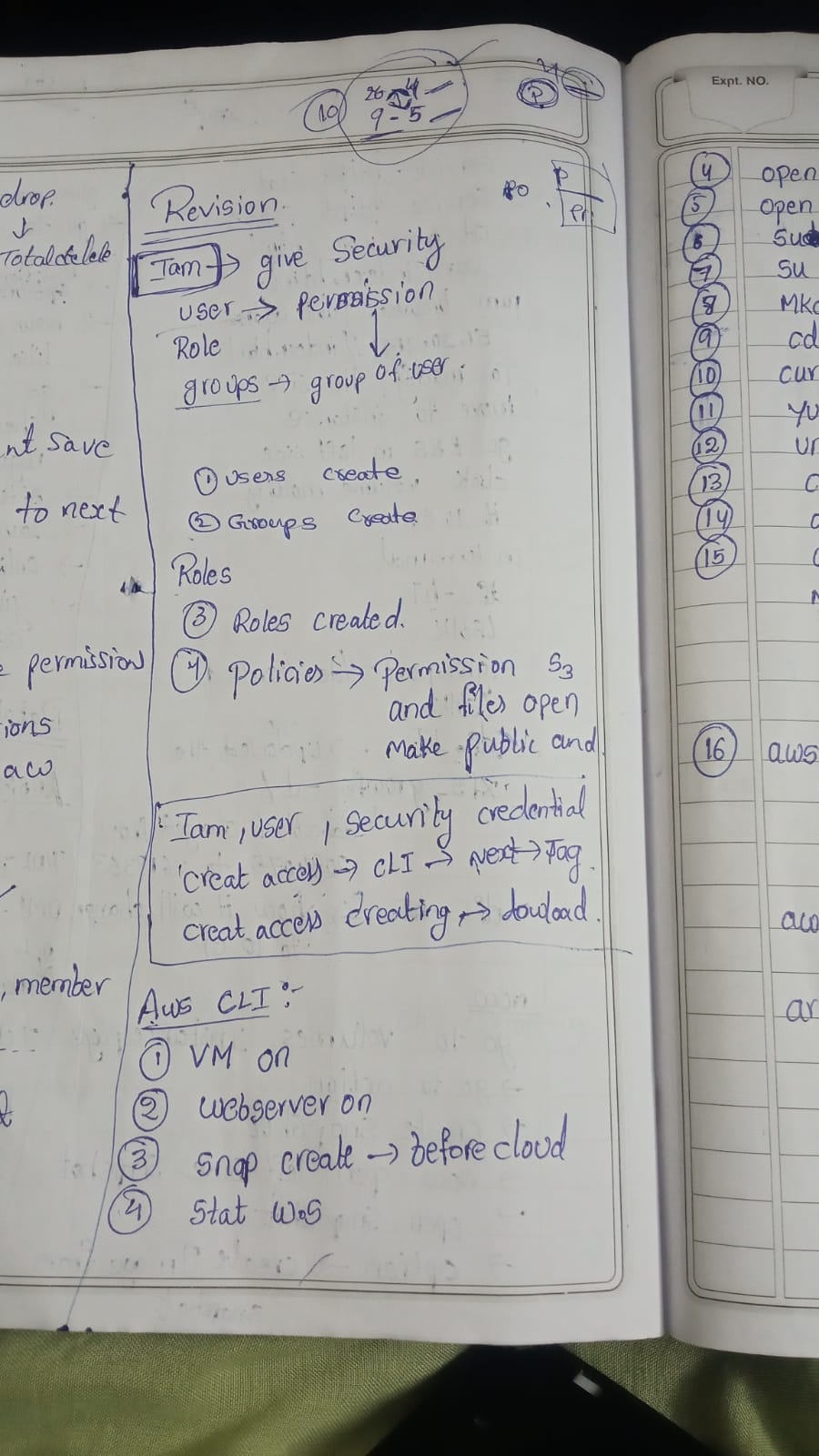
1. **General Purpose SSD:two types gp2,gp3**
2. **(gp3)**
   * **Performance**:
     + **IOPS**: Supports up to 16,000 IOPS per volume.
     + **Throughput**: Supports up to 1,000 MBps of throughput per volume.
   * **Use Cases**: Suitable for a wide range of workloads, including boot volumes, small to medium-sized databases, and development and test environments.
   * **Benefits**:
     + **Cost-Effective**: Provides a balance of performance and cost.
     + **Consistent Performance**: Designed to handle mixed workloads with predictable performance.
3. **Provisioned IOPS SSD :2 types**
4. **(io2):low latency ,high performance**
   * **Performance**:
     + **IOPS**: Can be provisioned up to 64,000 IOPS per volume.
     + **Throughput**: Supports up to 1,000 MBps of throughput per volume.
   * **Use Cases**: Ideal for mission-critical applications requiring high performance, such as large relational or NoSQL databases, and workloads demanding high and consistent IOPS.
   * **Benefits**:
     + **High Performance**: Provides very high IOPS and low-latency performance.
     + **Durability**: Designed for applications that require both high performance and high durability.
5. **Provisioned IOPS SSD (io1):demanding the work loads**
   * **Performance**:
     + **IOPS**: Can be provisioned up to 32,000 IOPS per volume.
     + **Throughput**: Supports up to 320 MBps of throughput per volume.
   * **Use Cases**: Suitable for large-scale, performance-sensitive applications like enterprise databases and critical applications requiring consistent high performance.
   * **Benefits**:
     + **High IOPS**: Ensures consistent, high performance for demanding workloads.
     + **Customizable Performance**: IOPS can be provisioned independently of volume size.

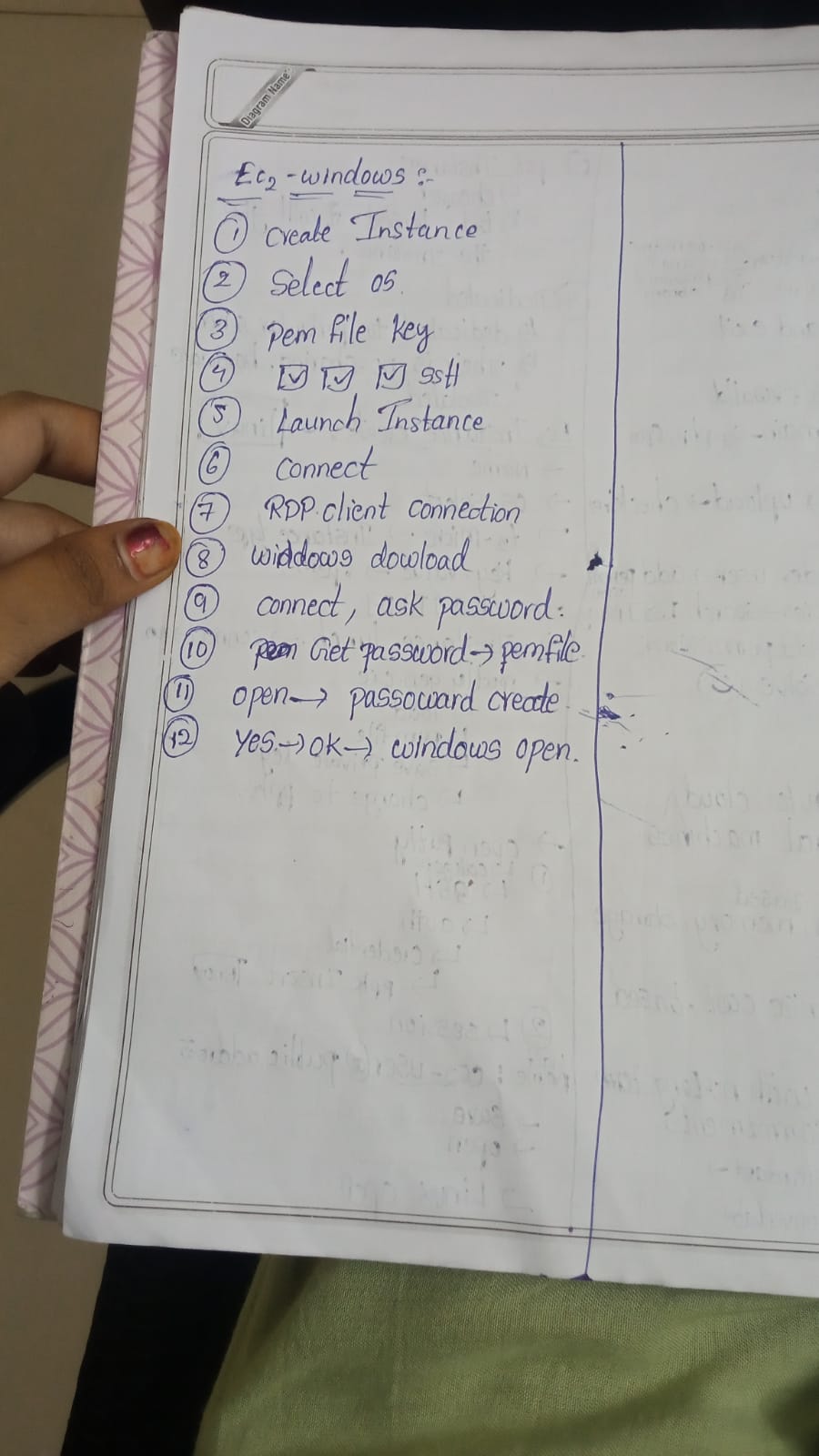
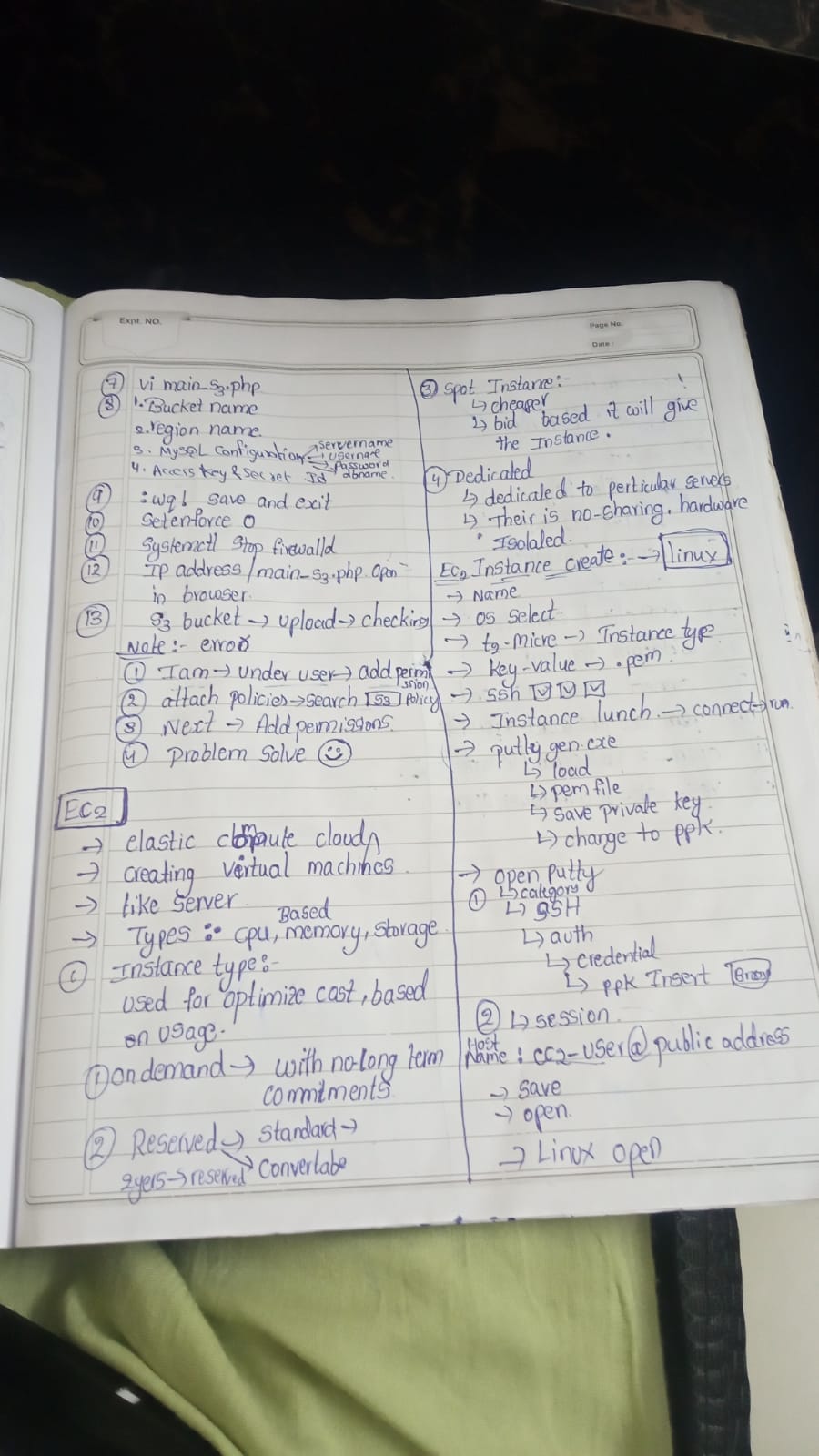
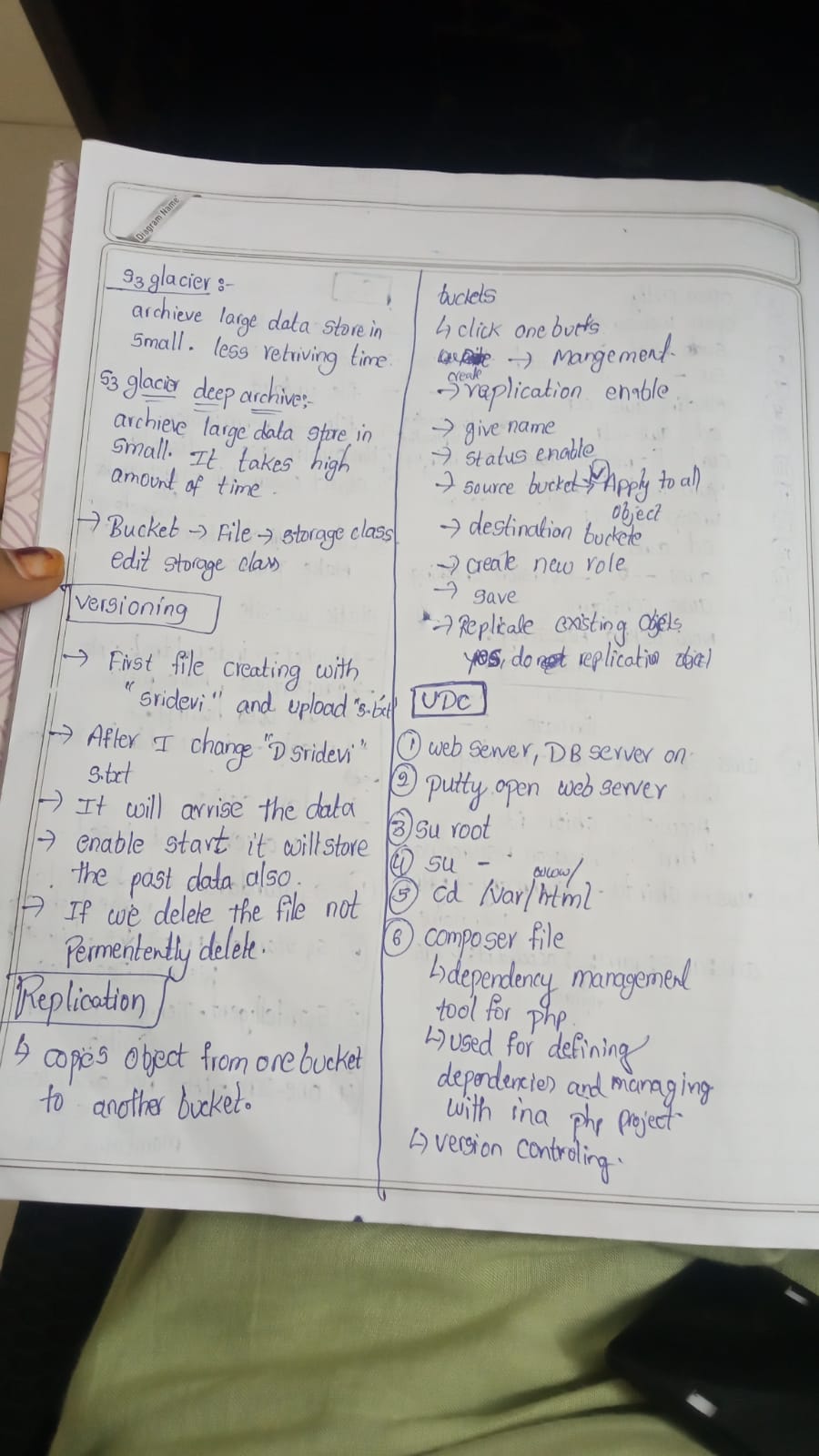
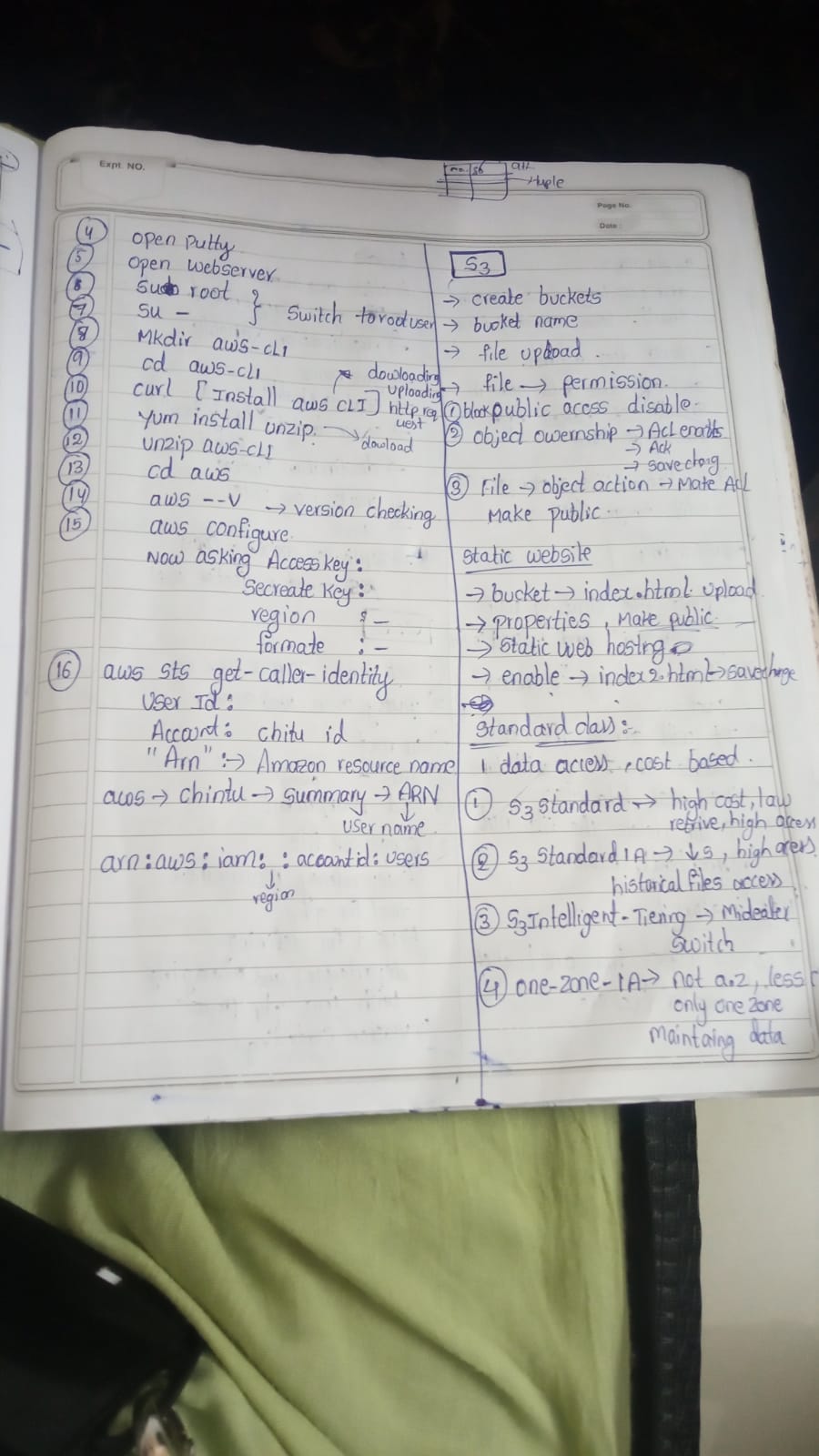
**EBS HDD Volume Types**

1. **Throughput Optimized HDD (st1)** **Performance**:
   * + **IOPS**: Provides up to 500 IOPS per volume.
     + **Throughput**: Supports up to 500 MBps of throughput per volume.
   * **Use Cases**: Designed for applications that require high throughput for large, sequential data operations. Commonly used for big data, data warehousing, and log processing.
   * **Benefits**:
     + **High Throughput**: Excellent for workloads that need to process large amounts of data quickly, such as data analytics or streaming data applications.
     + **Cost-Effective**: Lower cost per GB compared to SSD volumes, making it suitable for high-capacity, lower-cost storage needs.
2. **Cold HDD (sc1)**
   * **Performance**:
     + **IOPS**: Provides up to 250 IOPS per volume.
     + **Throughput**: Supports up to 250 MBps of throughput per volume.
   * **Use Cases**: Ideal for infrequently accessed data where throughput is important but high performance is not required. Examples include long-term archival storage, cold data storage, and backups.
   * **Benefits**:
     + **Low Cost**: Very economical for large volumes of data that are accessed infrequently.
     + **Suitable for Large Data Sets**: Provides a cost-effective way to store large amounts of data that don’t need high-speed access.

**Comparison of HDD Types**

* **Performance**:
  + **st1**: Offers higher throughput (up to 500 MBps) and more IOPS (up to 500) compared to **sc1**. Better suited for applications with higher performance requirements.
  + **sc1**: Lower throughput and IOPS, making it suitable for applications with lower performance needs but large data storage requirements.
* **Cost**:
  + **st1**: More expensive than **sc1** but still cheaper than SSD volumes. Provides a good balance of performance and cost.
  + **sc1**: The most cost-effective option for large volumes of infrequently accessed data. Lower cost per GB compared to **st1**.
* **Use Cases**:
  + **st1**: Best for workloads requiring high throughput and handling large-scale, sequential data operations. Examples include big data applications and large-scale data processing tasks.
  + **sc1**: Best for long-term storage and archiving where data is rarely accessed but needs to be stored cost-effectively. Suitable for backups and cold data storage.





**mysql**

**SQL (Structured Query Language)**

**What is SQL?**

SQL (Structured Query Language) is a standard programming language used to manage and manipulate relational databases. It allows you to perform various tasks such as:

* **Querying data:** Retrieving specific information from a database.
* **Inserting data:** Adding new records into a database.
* **Updating data:** Modifying existing records.
* **Deleting data:** Removing records from a database.
* **Creating and modifying database structures:** Defining tables, indexes, and constraints

**RDBMS (Relational Database Management System)**

**What is an RDBMS?**

An RDBMS (Relational Database Management System) is a type of database management system that stores data in a structured format, using rows and columns. It organizes data into tables, which can be linked—or related—based on common data attributes.

**Key Features of RDBMS:**

* **Tables:** Data is organized into tables. Each table has rows (records) and columns (attributes).
* **Relationships:** Tables can be related to one another through primary and foreign keys. A primary key uniquely identifies each record in a table, while a foreign key establishes a link between tables.
* **ACID Properties:** RDBMSs ensure reliable transactions using ACID (Atomicity, Consistency, Isolation, Durability) properties.
* **SQL Support:** RDBMSs use SQL to manage and query the data.

### 1. DDL - Data Definition Language

DDL commands are used to define or alter the structure of database objects, such as tables and schemas. These commands manage the schema of the database.

**1. CREATE**

The CREATE statement is used to create new tables, views, indexes, or databases.

**Example:**

Creating a new table called employees:

CREATE TABLE employees (

employee\_id INT PRIMARY KEY,

name VARCHAR(100),

position VARCHAR(50),

salary DECIMAL(10, 2)

);

**2. ALTER**

The ALTER statement is used to modify an existing database object, such as adding a column to a table or changing its structure.

**Example:**

Adding a new column hire\_date to the employees table:

ALTER TABLE employees

ADD hire\_date DATE;

To see the updated table structure, you would use a command like:

DESCRIBE employees;

**3.Drop:**

DROP TABLE employees;

**Output:**

No direct output. This command deletes the employees table and all its data.

To confirm the table is dropped, you would attempt to describe the table:

DESCRIBE employees;

**Output:**

Error: ERROR 1146 (42S02): Table 'database\_name.employees' doesn't exist

**2. DML - Data Manipulation Language**

**SELECT**

**SQL Command:**

SELECT \* FROM employees;

The output would be:

| **employee\_id** | **name** | **position** | **salary** | **hire\_date** |
| --- | --- | --- | --- | --- |
| 1 | Alice Johnson | Developer | 75000 | 2024-07-31 |
| 2 | Bob Smith | Salesman | 68000 | 2024-07-30 |

**INSERT**

**SQL Command:**

INSERT INTO employees (employee\_id, name, position, salary, hire\_date)

VALUES (3, 'Charlie Brown', 'Manager', 90000, '2024-07-31');

SELECT \* FROM employees;

**Output:**

| **employee\_id** | **name** | **position** | **salary** | **hire\_date** |
| --- | --- | --- | --- | --- |
| 1 | Alice Johnson | Developer | 75000 | 2024-07-31 |
| 2 | Bob Smith | Salesman | 68000 | 2024-07-30 |
| 3 | Charlie Brown | Manager | 90000 | 2024-07-31 |

**UPDATE**

**SQL Command:**

UPDATE employees

SET salary = 80000

WHERE employee\_id = 1;

**Output:**

| **employee\_id** | **name** | **position** | **salary** | **hire\_date** |
| --- | --- | --- | --- | --- |
| 1 | Alice Johnson | Developer | 80000 | 2024-07-31 |
| 2 | Bob Smith | Salesman | 68000 | 2024-07-30 |
| 3 | Charlie Brown | Manager | 90000 | 2024-07-31 |

**DELETE**

**SQL Command:**

DELETE FROM employees

WHERE employee\_id = 1;

**Output:**

| **employee\_id** | **name** | **position** | **salary** | **hire\_date** |
| --- | --- | --- | --- | --- |
| 2 | Bob Smith | Salesman | 68000 | 2024-07-30 |
| 3 | Charlie Brown | Manager | 90000 | 2024-07-31 |

**3. DCL - Data Control Language**

**GRANT: Purpose: The GRANT statement is used to give specific permissions to users or roles. Permissions can include the ability to read, write, or modify data.**

GRANT privilege\_type ON object TO user\_or\_role;

* **privilege\_type**: The type of permission being granted (e.g., SELECT, INSERT, UPDATE, DELETE).
* **object**: The database object (e.g., table, view) on which the permissions are granted.
* **user\_or\_role**: The user or role to which the permissions are being granted.

GRANT SELECT, INSERT ON employees TO 'john\_doe';

**Explanation:**

* **SELECT**: Allows john\_doe to retrieve data from the employees table.
* **INSERT**: Allows john\_doe to add new records to the employees table.

**Verification:**

To see what privileges a user has, you can query the database system’s information schema (varies by database system):

SHOW GRANTS FOR 'john\_doe';

**Example2:**

Assume we have a table called employees and a user named john\_doe. We want to give john\_doe the ability to read data (SELECT) and add new data (INSERT) to the employees table.

**SQL Command:**

**SQL Command:**

GRANT SELECT, INSERT ON employees TO 'john\_doe';

**Output:**

No direct output. This command grants john\_doe the ability to select and insert data into the employees table.

To verify, you would typically check the user permissions through database-specific views or system tables.

**REVOKE**

**SQL Command:**

REVOKE INSERT ON employees FROM 'john\_doe';

**Output:**

No direct output. This command revokes the INSERT privilege on the employees table from john\_doe.

**4. TCL - Transaction Control Language**

**COMMIT**

**SQL Command:**

COMMIT;

**0utput:**

No direct output. This command saves all changes made in the current transaction.

**ROLLBACK**

**SQL Command:**

ROLLBACK;

**Output:**

No direct output. This command undoes all changes made in the current transaction since the last COMMIT or ROLLBACK.

**SAVEPOINT**

**SQL Command:**

SAVEPOINT savepoint1;

**Output:**

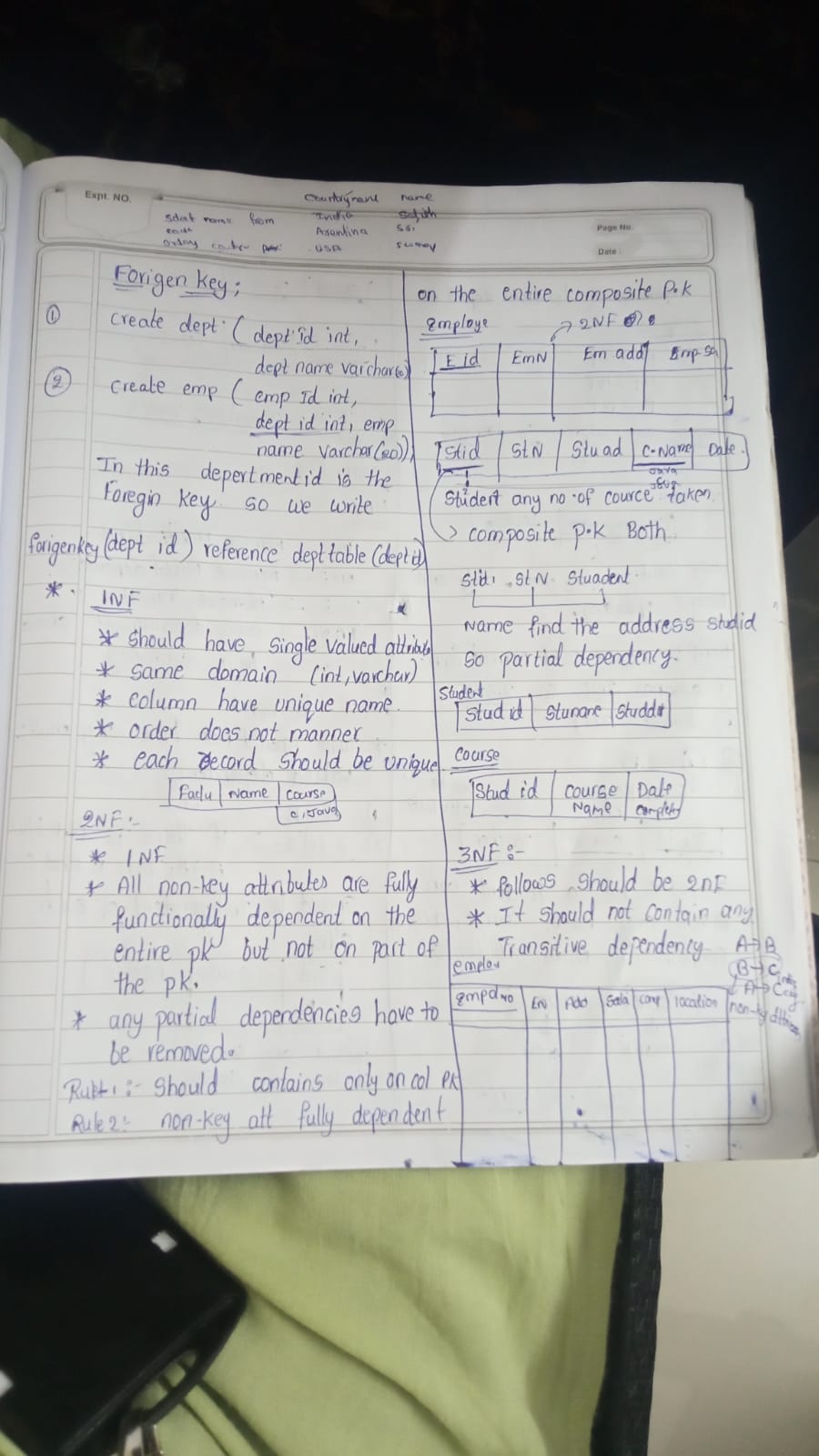
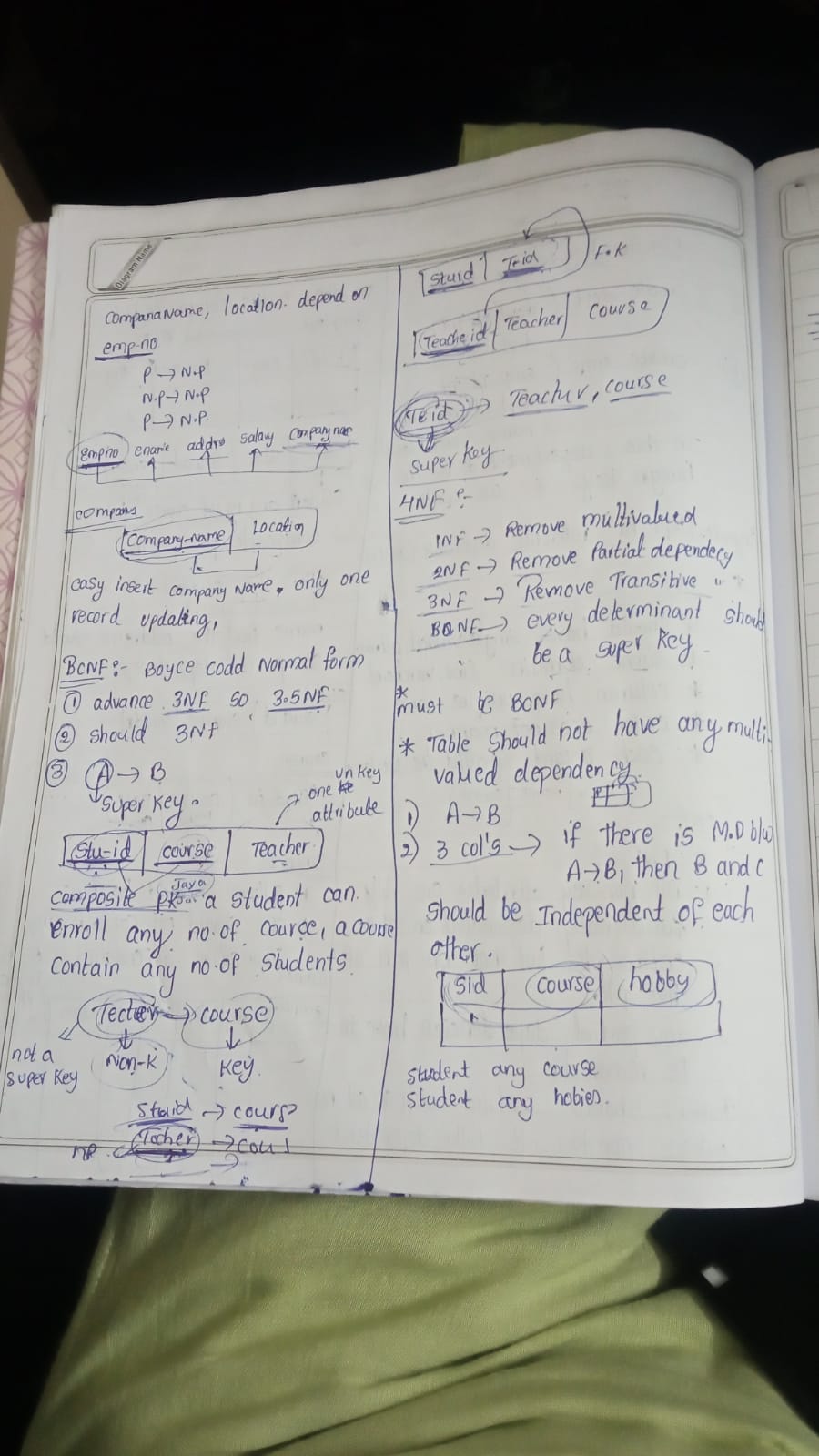
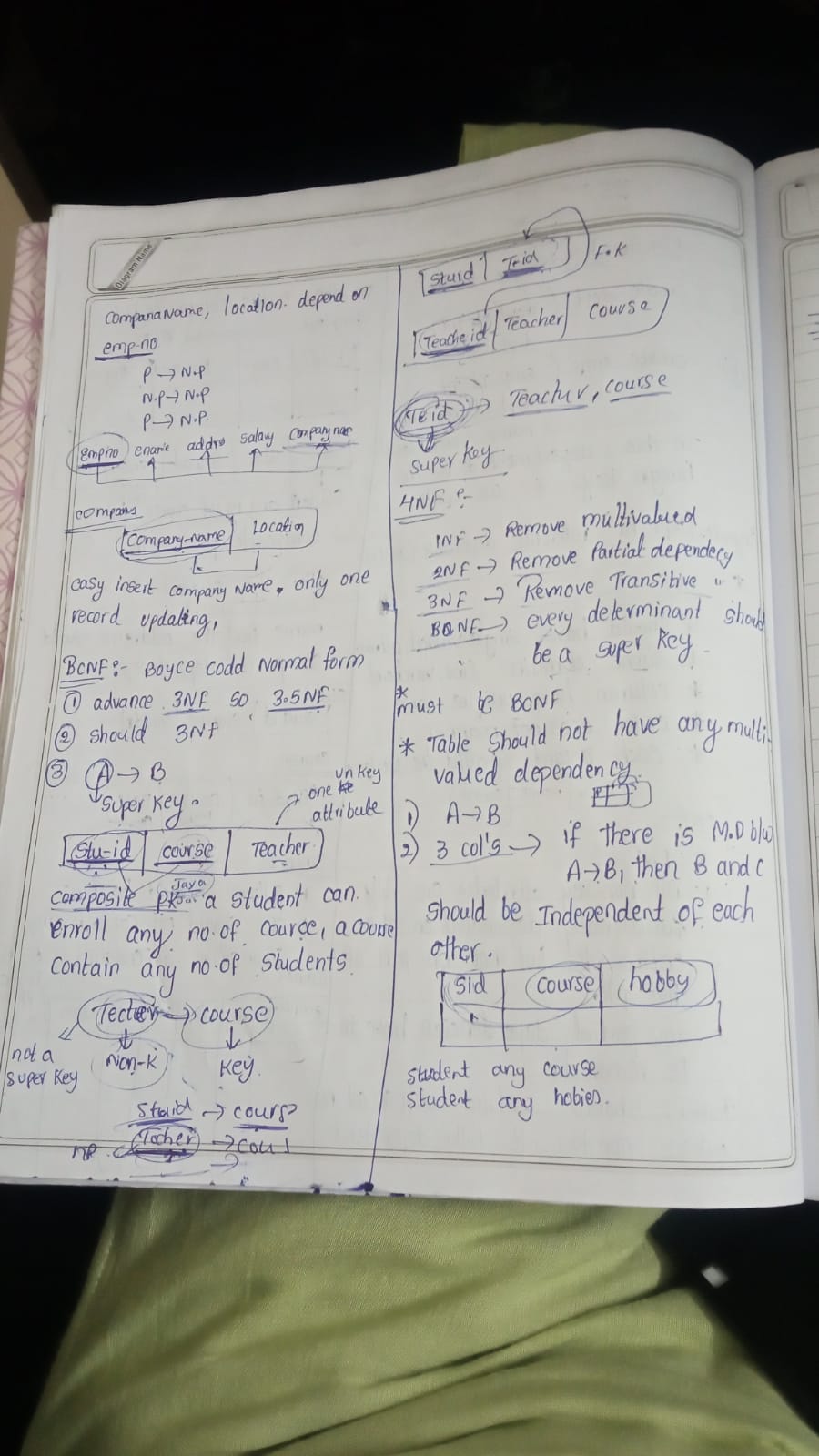
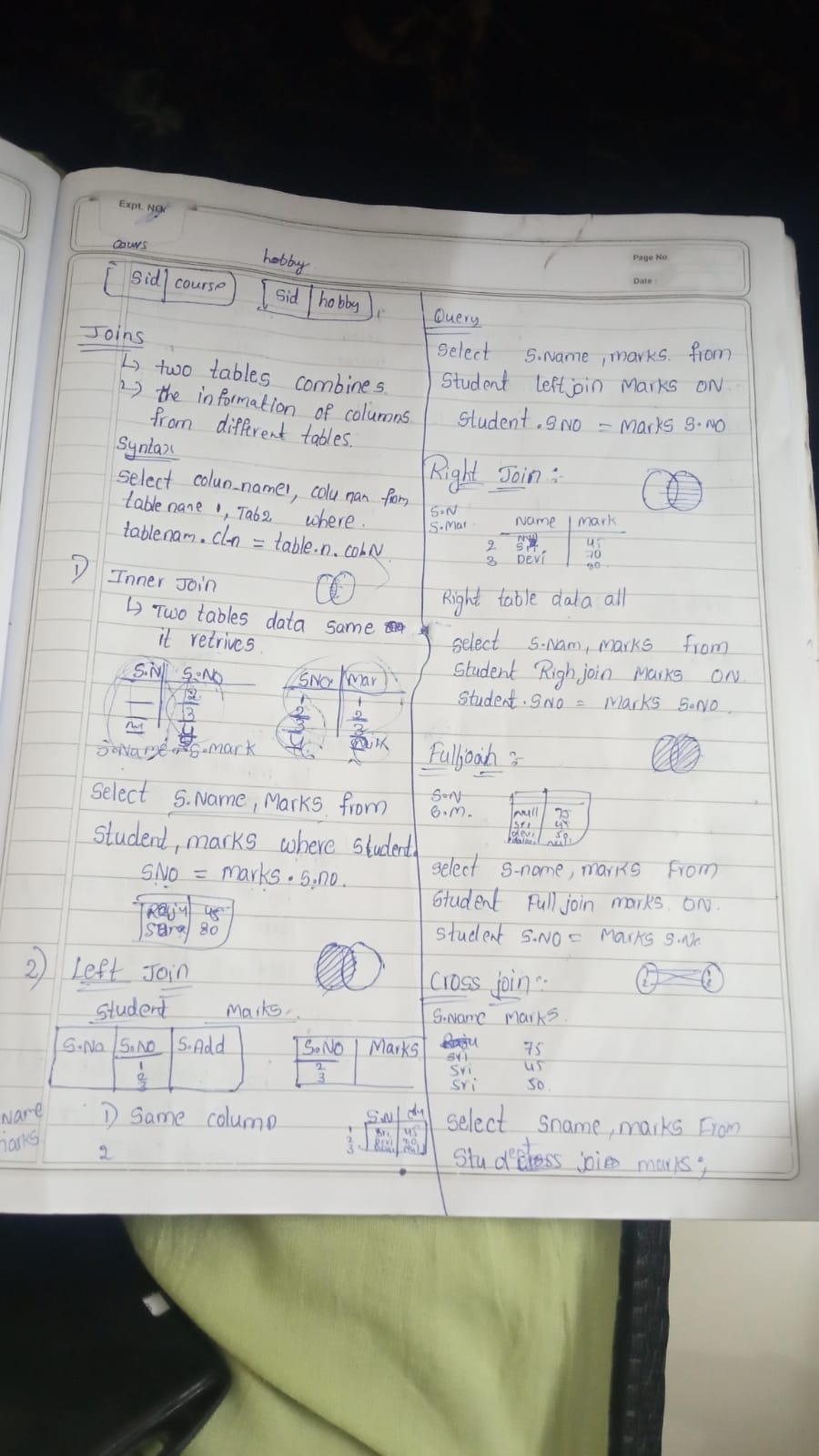
No direct output. This command sets a savepoint named savepoint1 within the current transaction.

To roll back to this savepoint:

ROLLBACK TO SAVEPOINT savepoint1;

**Output:**

No direct output. This command undoes changes made after the savepoint1, but keeps the changes made before it.



 **INNER JOIN**

* **Purpose:** Returns rows with matching values in both tables.
* **Example:** Retrieve employees and their department names where there is a match in both the employees and departments tables.

 **LEFT JOIN (LEFT OUTER JOIN)**

SELECT employees.name, departments.department\_name

FROM employees

LEFT JOIN departments

ON employees.department\_id = departments.department\_id;

* **Purpose:** Returns all rows from the left table, and matched rows from the right table. If there is no match, NULL values are returned for columns from the right table.
* **Example:** Get all employees and their department names, including employees who may not be assigned to any department.

 **RIGHT JOIN (RIGHT OUTER JOIN)**

* **Purpose:** Returns all rows from the right table, and matched rows from the left table. If there is no match, NULL values are returned for columns from the left table.
* **Example:** List all departments and any employees assigned to them, including departments that may not have any employees.

 **FULL JOIN (FULL OUTER JOIN)**

* **Purpose:** Returns all rows when there is a match in either the left or right table. If there is no match, NULL values are returned for columns from the table without a match.
* **Example:** Show all employees and departments, with NULL values for departments without employees and employees not assigned to any department.

 **CROSS JOIN**

* **Purpose:** Returns the Cartesian product of the two tables, pairing each row from the first table with every row from the second table.
* **Example:** Combine each employee with every department, regardless of actual assignment.

 **SELF JOIN**

* **Purpose:** Joins a table with itself to relate rows within the same table.
* **Example:** List employees alongside their managers by joining the employees table with itself based on manager\_id.

**What is Amazon EBS?**

**Amazon EBS (Elastic Block Store)** is a scalable and high-performance block storage service provided by AWS. It is used to store data for Amazon EC2 (Elastic Compute Cloud) instances and provides persistent storage that remains available even when the instance is stopped or terminated.

**Why Use Amazon EBS?**

1. **Persistent Storage**: EBS volumes persist beyond the lifetime of individual EC2 instances. Data remains intact even if you stop or terminate your instance.
2. **Scalability**: You can easily scale your storage up or down by creating new volumes or resizing existing ones.
3. **Performance**: Offers various volume types optimized for different use cases, such as general-purpose SSDs, provisioned IOPS SSDs, and magnetic disks.
4. **Backup and Recovery**: You can take snapshots of EBS volumes for backup and disaster recovery. Snapshots are stored in Amazon S3 and can be used to create new EBS volumes.

**What is the Purpose of Amazon EBS?**

* **Root Volumes**: EBS is commonly used as the root volume for an EC2 instance, containing the operating system and boot files.
* **Data Storage**: Provides additional storage for applications, databases, logs, and other data.
* **Performance**: Ensures high-performance storage for applications that require fast access to data, such as databases and data processing applications.
* **Backups**: Facilitates backup and disaster recovery strategies by creating snapshots.

**How to Use Amazon EBS with a Simple Example**

Here’s a step-by-step guide to creating and using an Amazon EBS volume with an EC2 instance:

**1. Create an EBS Volume**

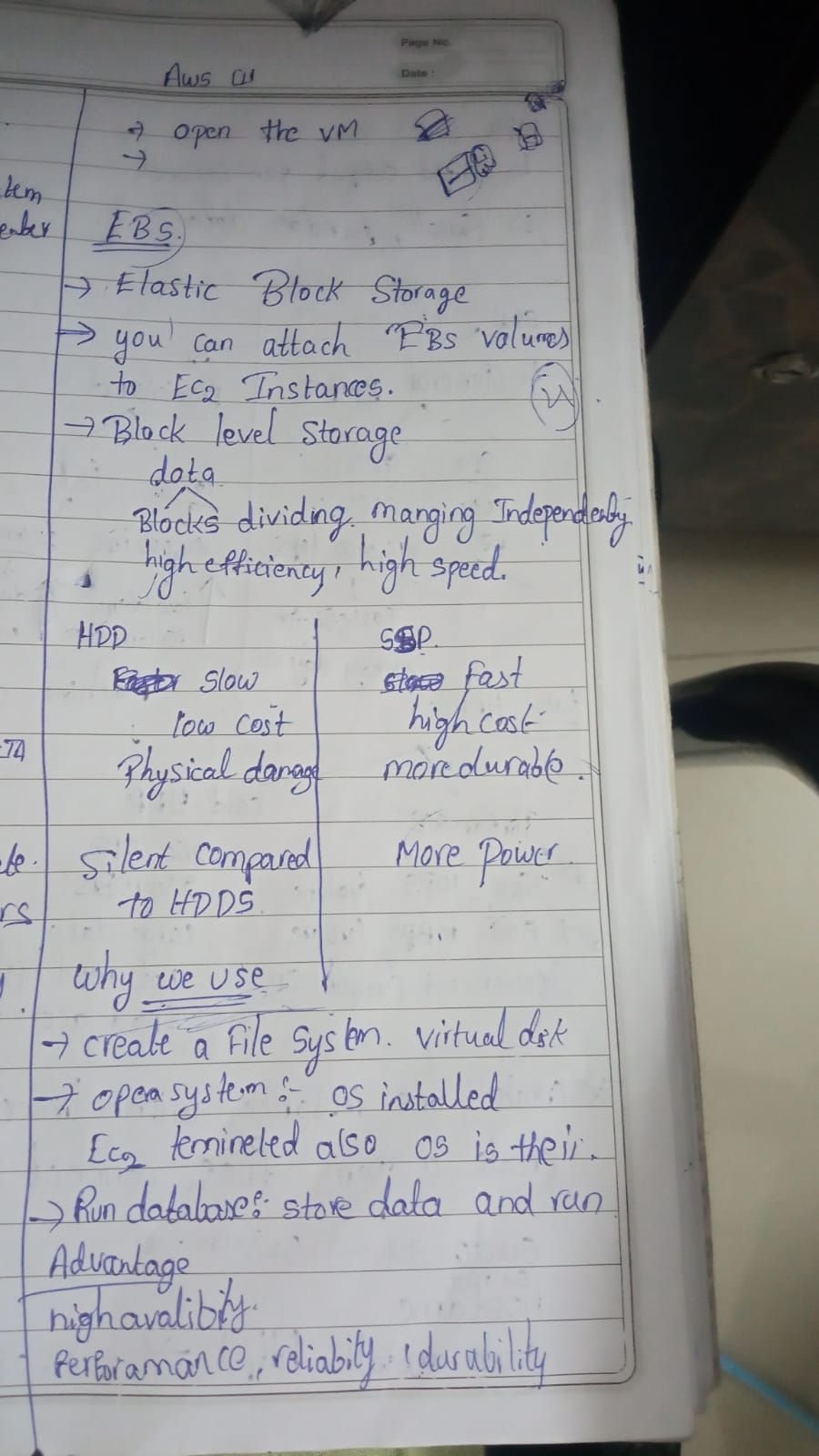
1. **Open AWS Management Console**:
   * Go to the **EC2** dashboard.
2. **Create a Volume**:
   * Select **Volumes** under **Elastic Block Store** in the left-hand menu.
   * Click **Create Volume**.
   * Configure the volume:
     + **Size**: Specify the size of the volume (e.g., 10 GB).
     + **Type**: Choose a volume type (e.g., gp3 for general-purpose SSD).
     + **Availability Zone**: Choose the same availability zone as your EC2 instance.
3. **Create the Volume**:
   * Click **Create Volume** to complete the process.

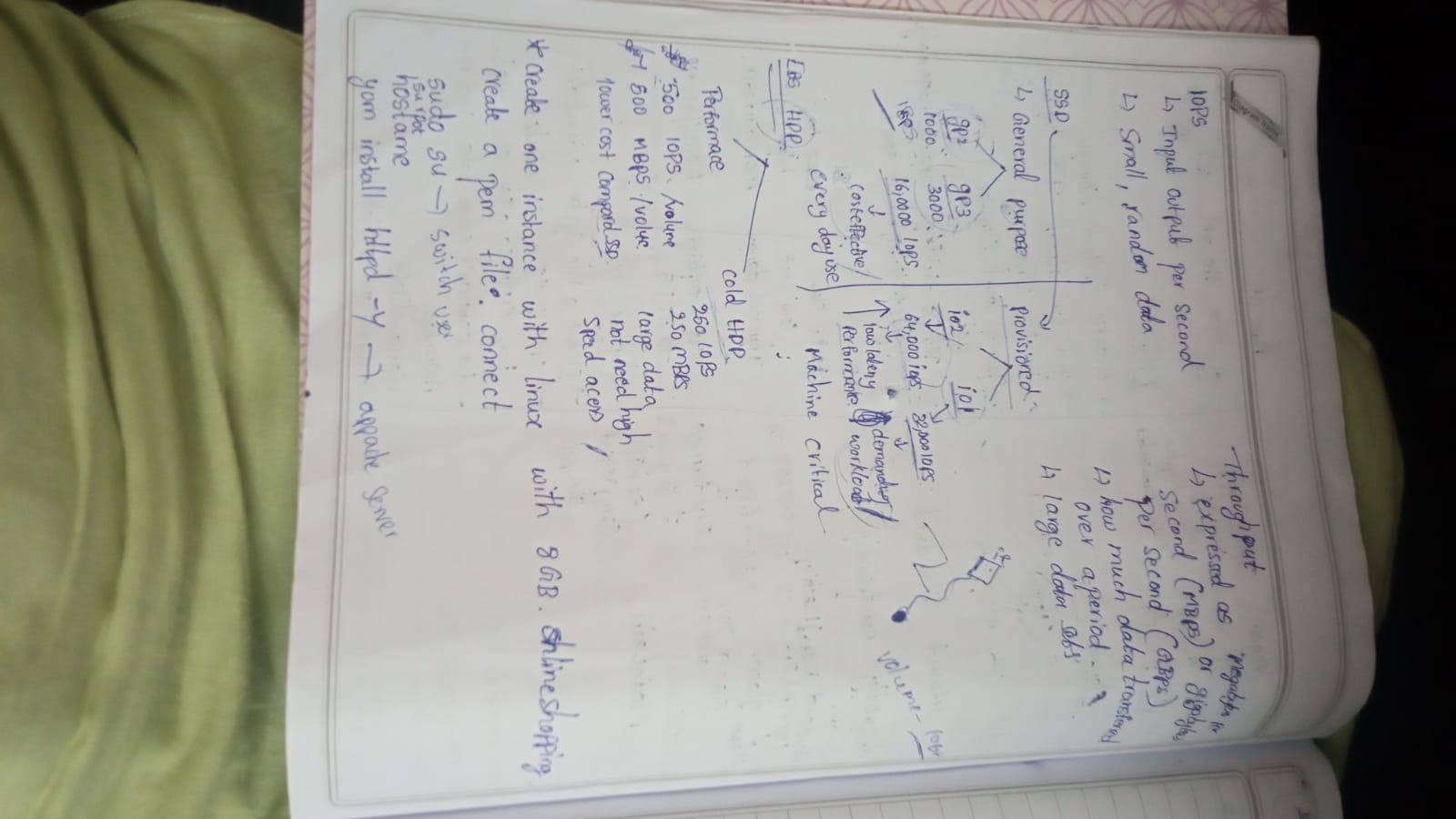
**2. Attach the EBS Volume to an EC2 Instance**

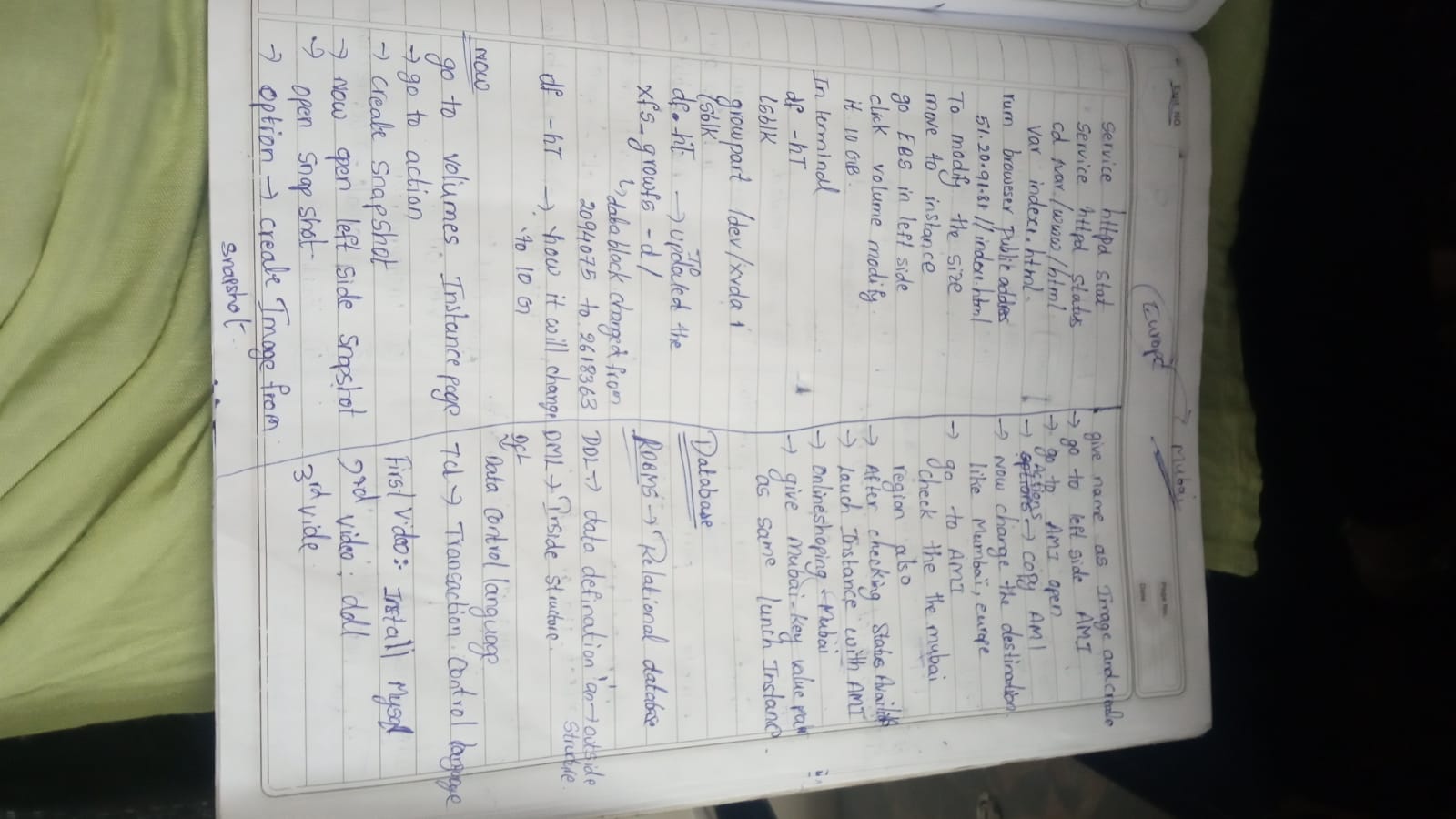
1. **Select the Volume**:
   * In the **Volumes** section, select the volume you just created.
2. **Attach the Volume**:
   * Click **Actions** and then **Attach Volume**.
   * Select the EC2 instance you want to attach the volume to.
   * Specify the device name (e.g., /dev/xvdf).
   * Click **Attach Volume**.

**3. Prepare the EBS Volume on Your EC2 Instance**

1. **SSH into Your EC2 Instance**:
   * Connect to your EC2 instance using SSH.







**What is Amazon EFS?**

**Amazon EFS (Elastic File System)** is a scalable, fully managed network file system that you can use with AWS Cloud services and on-premises resources. It provides a simple, scalable, and elastic file storage solution that can be used across multiple EC2 instances simultaneously.

**Why Use Amazon EFS?**

1. **Scalability**: Automatically scales up or down based on the amount of data stored. No need to manually provision or manage storage capacity.
2. **Accessibility**: Allows multiple EC2 instances to access the same file system concurrently, making it ideal for applications that require shared access to data.
3. **High Availability**: Provides high durability and availability by storing data across multiple Availability Zones.
4. **Managed Service**: AWS handles the infrastructure management, including scaling, patching, and backups, allowing you to focus on your applications.

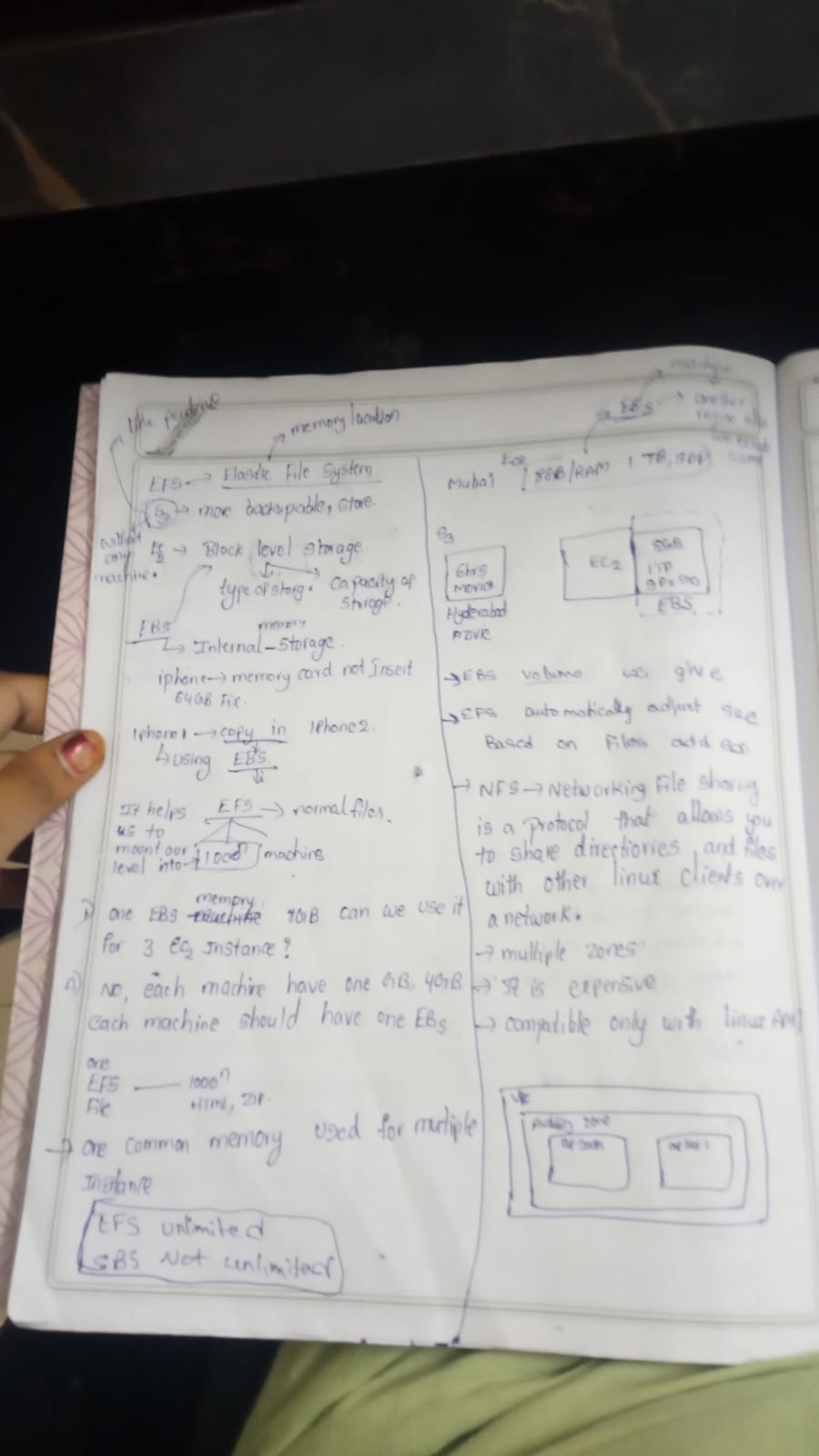
**When to Use Amazon EFS?**

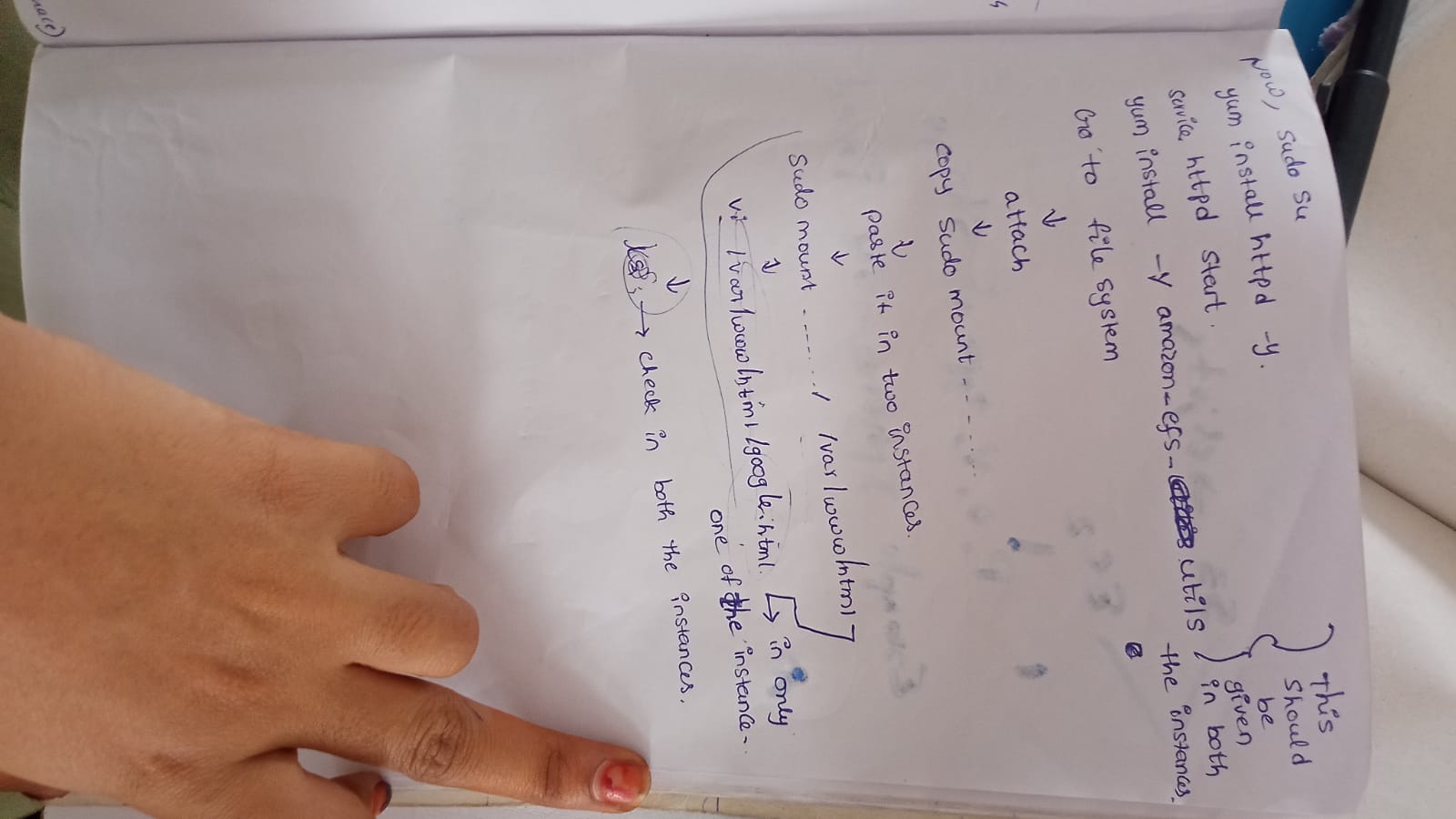
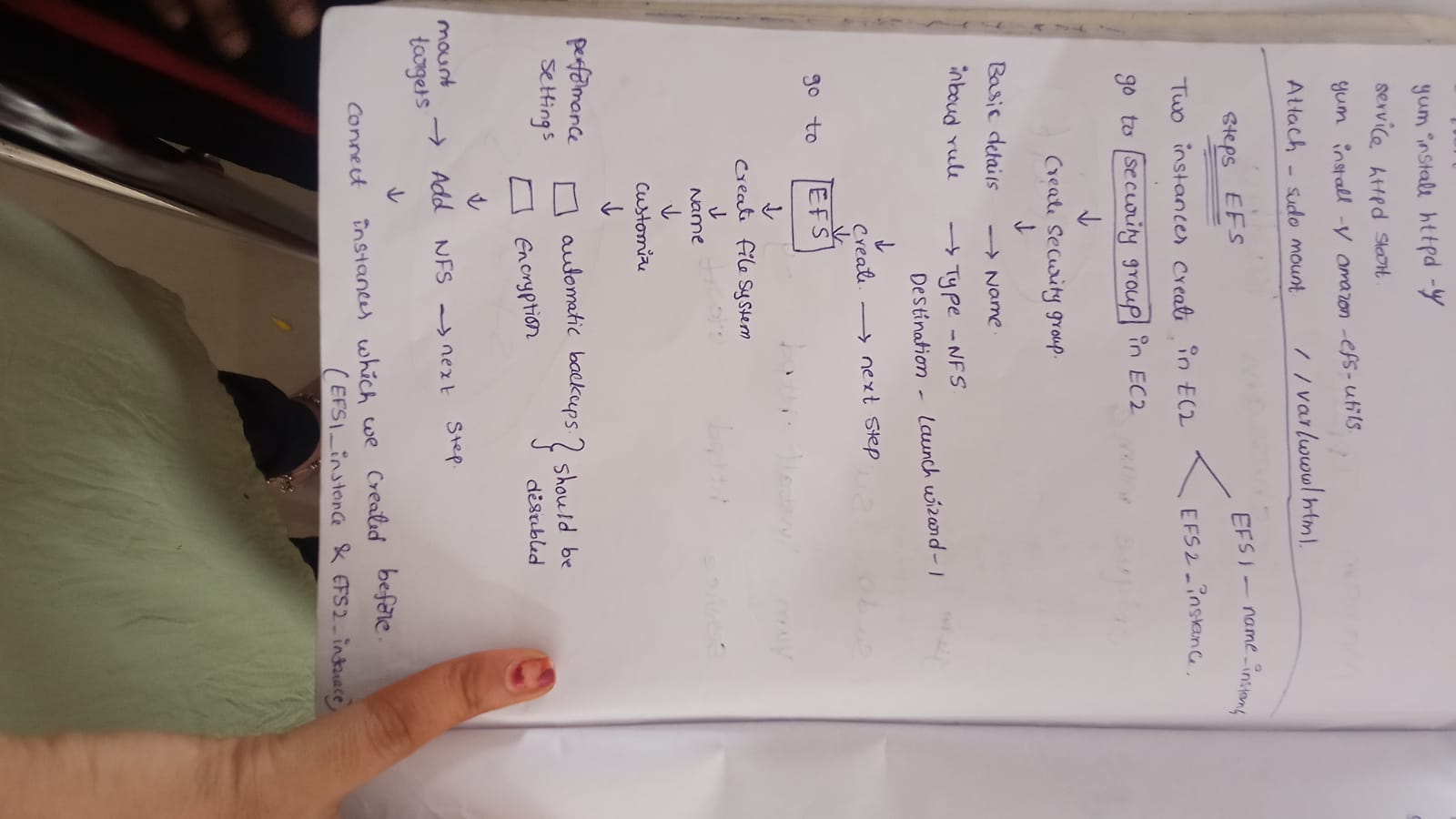
1. **Shared Storage**: When you need a file system that can be accessed concurrently by multiple EC2 instances.
2. **Application Data**: For applications that require a shared data repository, such as content management systems, big data applications, or web servers with shared files.
3. **Home Directories**: For storing user home directories that need to be accessed from multiple instances.
4. **Backup and Recovery**: For applications that require high availability and automatic backup.

**How to Use Amazon EFS in AWS**

Here’s a simple step-by-step guide to using Amazon EFS:

1. **Create an EFS File System**
   * **Step 1**: Open the AWS Management Console.
   * **Step 2**: Go to the **EFS** dashboard.
   * **Step 3**: Click **Create file system**.
   * **Step 4**: Configure the file system settings:
     + **VPC**: Select the VPC where your EC2 instances are located.
     + **Availability Zones**: Choose the availability zones to mount the file system.
     + **Performance Mode**: Choose between **General Purpose** (default) or **Max I/O** depending on your performance needs.
   * **Step 5**: Review and create the file system.
2. **Configure Security Groups and Mount Targets**
   * **Step 1**: Go to the **Network** section of the EFS file system configuration.
   * **Step 2**: Ensure that the file system has mount targets in the same VPC and availability zones as your EC2 instances.
   * **Step 3**: Configure security groups to allow access to the EFS file system from your EC2 instances (typically TCP port 2049 for NFS).



****

**What is a Load Balancer?**

**Definition**: A load balancer is a device or service that distributes incoming network or application traffic across multiple servers. The goal is to ensure that no single server becomes overwhelmed with too much traffic, which helps in maintaining performance and reliability.

**Why Use a Load Balancer?**

1. **Distribute Traffic**: It evenly distributes incoming traffic across multiple servers, preventing any single server from becoming a bottleneck.
2. **Improve Reliability**: If one server fails, the load balancer redirects traffic to the remaining healthy servers, ensuring that the application remains available.
3. **Enhance Performance**: By balancing the load, you ensure that each server handles a manageable amount of traffic, which can improve response times and overall performance.
4. **Scalability**: Easily add or remove servers from the pool to handle changes in traffic volume, without affecting the end-user experience.

**How to Use a Load Balancer**

**In AWS**, the primary service for load balancing is **Elastic Load Balancing (ELB)**. Here’s a step-by-step guide on how to use it:

1. **Create a Load Balancer**
   * **Step 1**: Go to the AWS Management Console.
   * **Step 2**: Navigate to the "EC2" dashboard.
   * **Step 3**: Select "Load Balancers" from the menu.
   * **Step 4**: Click "Create Load Balancer" and choose the type (Application Load Balancer, Network Load Balancer, or Classic Load Balancer).
2. **Configure the Load Balancer**
   * **Name**: Give your load balancer a name.
   * **Scheme**: Choose whether it’s internet-facing or internal.
   * **Listeners**: Define the protocol and port (e.g., HTTP on port 80).
   * **Availability Zones**: Select the availability zones where your load balancer will be deployed.
3. **Configure Security Settings**
   * **Security Groups**: Choose or create security groups to control access to the load balancer.
4. **Configure Target Groups**
   * **Target Group**: Create a target group that includes the EC2 instances or IP addresses that will receive the traffic.
   * **Health Checks**: Define health check settings to determine if the targets are healthy and can receive traffic.
5. **Review and Create**
   * **Review**: Check all the configurations.
   * **Create**: Click "Create Load Balancer" to complete the setup.
6. **Add Instances to Target Groups**
   * **Register Targets**: Go to the target group and register your EC2 instances.

**When to Use a Load Balancer**

1. **High Traffic Applications**: When your application receives a lot of traffic and you need to distribute the load across multiple servers to prevent any one server from becoming a bottleneck.
2. **High Availability**: To ensure that your application remains available even if one or more servers fail. The load balancer can automatically reroute traffic to healthy servers.
3. **Scalability Needs**: When you expect fluctuating traffic volumes and need to easily add or remove servers from the pool based on current demand.
4. **Maintenance**: When performing maintenance on one or more servers, the load balancer can direct traffic away from those servers to others that are operational.

**1. Health Checks**

**Definition**: Health Checks are used to monitor the health of your servers (or targets) to ensure they can handle incoming requests. The load balancer uses these checks to determine which servers are healthy and should receive traffic.

**How It Works**:

* **Health Check Configuration**: You define how the load balancer should check if a server is healthy (e.g., via HTTP requests).
* **Monitoring**: The load balancer periodically sends health check requests to the servers.
* **Routing**: If a server fails health checks, the load balancer stops sending traffic to that server until it passes the health checks again.

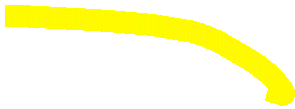
**Simple Example**: Imagine you have a website with 3 servers behind a load balancer. The load balancer performs health checks by making HTTP requests to each server’s health endpoint (like /health).

* **Healthy Server**: Responds with a 200 OK status code.
* **Unhealthy Server**: Responds with an error or doesn’t respond.

If one server fails the health check, the load balancer will stop sending traffic to it until it is healthy again.

**2. Listeners**

**Definition**: Listeners are the entry points for traffic into your load balancer. They define the protocol and port on which the load balancer listens for incoming requests.



**How It Works**:



* **Protocol and Port**: Listeners listen on specified protocols (HTTP, HTTPS, TCP) and ports (80, 443, etc.).
* **Forwarding Traffic**: Listeners forward incoming traffic to the appropriate target groups based on rules and configurations.

**Simple Example**: Imagine you have a web application running on port 80 (HTTP) and another service running on port 443 (HTTPS).

* **HTTP Listener**: Configured to listen on port 80. It handles incoming HTTP requests and forwards them to your web servers.
* **HTTPS Listener**: Configured to listen on port 443. It handles incoming HTTPS requests (encrypted) and forwards them to your servers after decrypting the traffic.

**3. Rules**

**Definition**: Rules define how the load balancer should route incoming traffic to different target groups based on conditions like URL paths or host headers.

**How It Works**:

* **Routing Logic**: Rules specify conditions that determine how requests should be routed.
* **Target Groups**: Requests matching specific rules are forwarded to designated target groups.

**Simple Example**: Let’s say you have a website with different services:

* **Service A**: Handles /images and /static requests.
* **Service B**: Handles all other requests (e.g., /api, /docs).

**Rules Configuration**:

* **Rule 1**: If the request URL path starts with /images or /static, route it to the "Images Service" target group.
* **Rule 2**: For all other requests, route them to the "Main Service" target group.

**Classic Load Balancer (CLB)**

**Definition**: The Classic Load Balancer is the original AWS load balancer, designed to handle both HTTP/HTTPS and TCP traffic. It is often used for legacy applications.

**Features**:

* **Traffic Distribution**: Can distribute traffic based on TCP or HTTP/HTTPS protocols.
* **Basic Functionality**: Provides basic load balancing capabilities.
* **Limited Features**: Lacks advanced features found in ALB and NLB.

**Best Use Case**:

* Simple applications that do not need advanced routing or high-performance features.

**Example**: You have a basic web application and you need to distribute HTTP requests across a few EC2 instances. CLB will balance the load evenly across those instances.

**2. Application Load Balancer (ALB)**

**Definition**: The Application Load Balancer is designed for advanced HTTP/HTTPS load balancing. It operates at the application layer (Layer 7) and offers more sophisticated routing capabilities.

**Features**:

* **Advanced Routing**: Supports URL-based routing, host-based routing, and WebSocket support.
* **Target Groups**: Allows you to route traffic to different target groups based on rules.
* **Layer 7**: Operates at the application layer, handling HTTP and HTTPS traffic.

**Best Use Case**:

* Modern web applications with complex routing needs, such as directing different URL paths to different services.

**Example**: For a website that needs to route requests to different microservices (e.g., /api routes to one service and /static routes to another), ALB can direct traffic based on URL paths.

**3. Network Load Balancer (NLB)**

**Definition**: The Network Load Balancer is designed for high-performance, TCP traffic. It operates at the transport layer (Layer 4) and is optimized for handling millions of requests per second with low latency.

**Features**:

* **High Performance**: Handles large amounts of TCP traffic with minimal latency.
* **Static IP**: Provides a static IP address for clients to connect to.
* **Layer 4**: Operates at the transport layer, handling TCP traffic.

**Best Use Case**:

* Applications requiring high-performance and low-latency TCP traffic handling, such as real-time gaming or financial transactions.

**Example**: If you have a financial application that needs to handle high-volume, low-latency transactions, NLB ensures that traffic is efficiently routed with minimal delay.

**4. Gateway Load Balancer (GWLB)**

**Definition**: The Gateway Load Balancer is designed for managing and scaling third-party virtual appliances, like firewalls or intrusion detection systems. It combines a transparent network gateway with load balancing.

**Features**:

* **Transparent Integration**: Provides a way to deploy and scale virtual appliances while distributing traffic.
* **Layer 3/4**: Operates at the network and transport layers.
* **Integration**: Ideal for integrating with third-party network appliances.

**Best Use Case**:

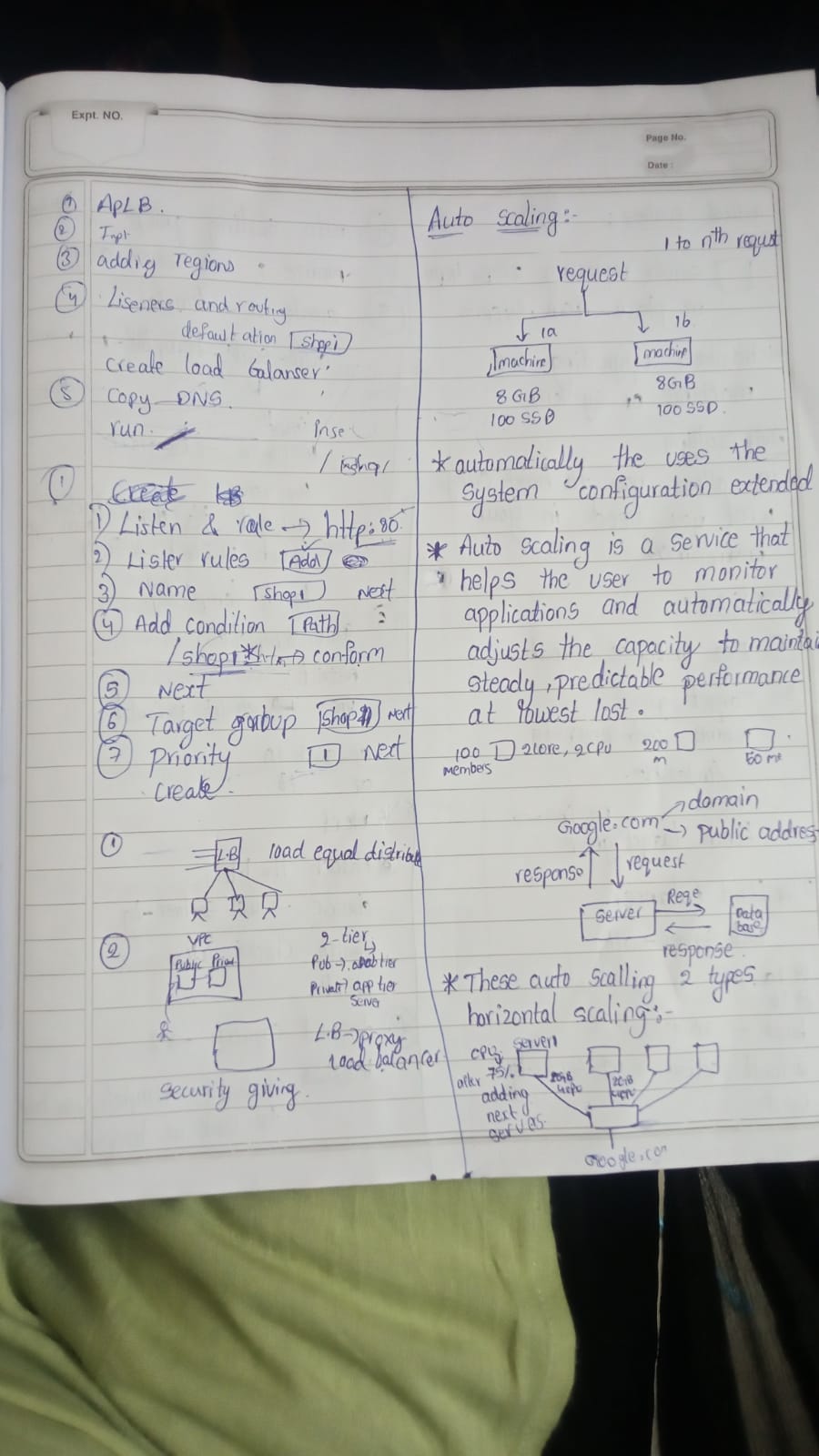
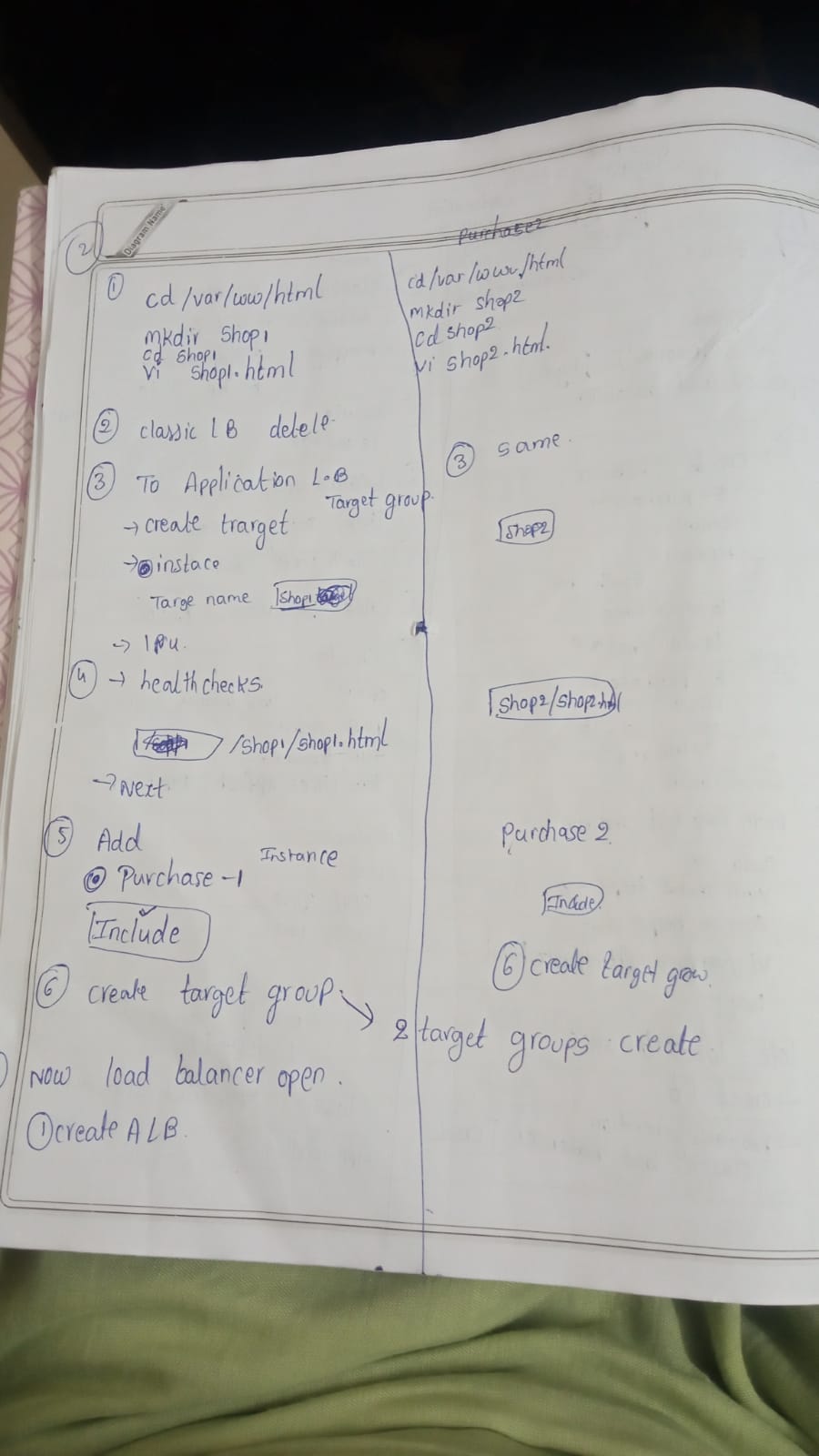
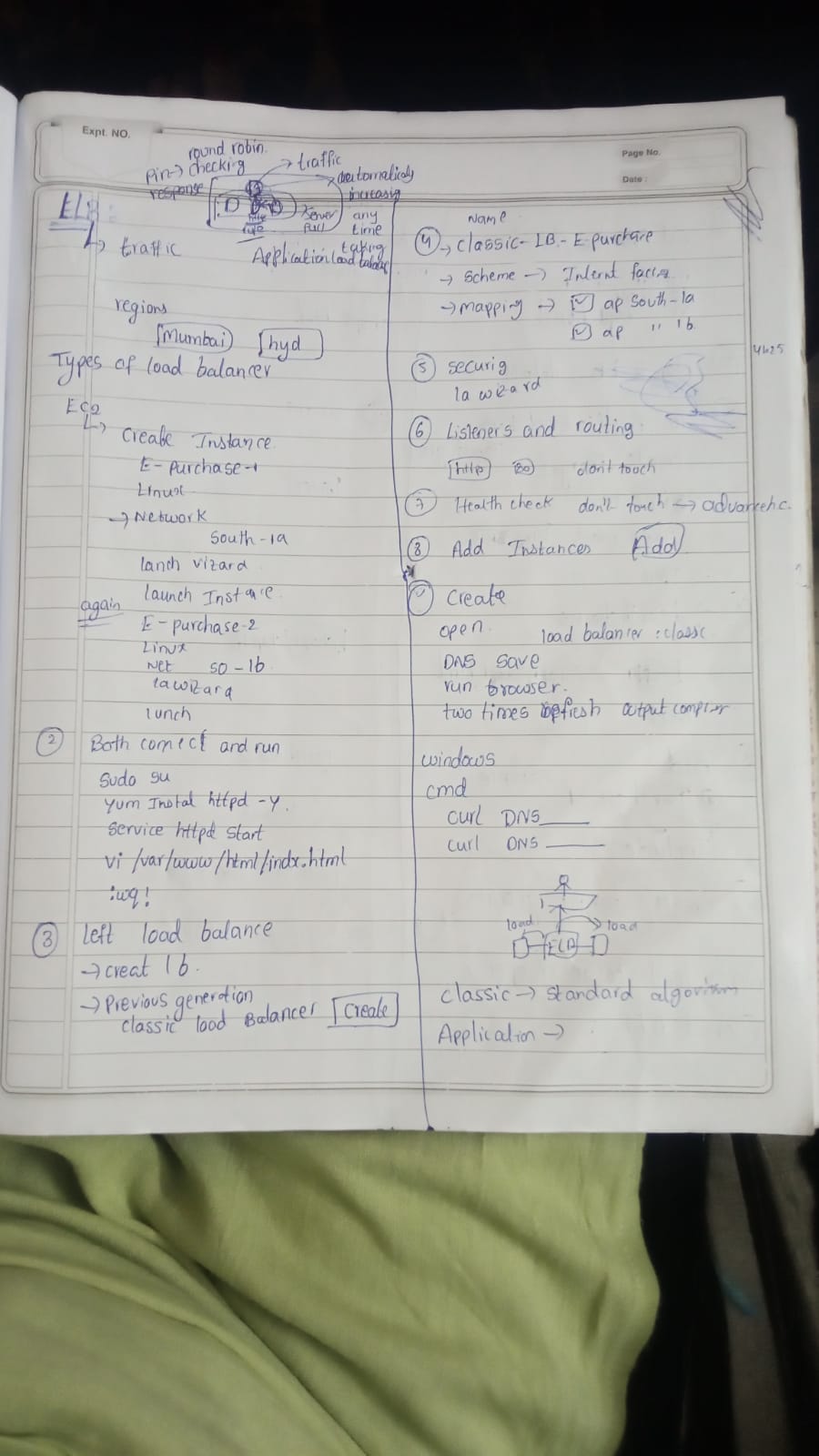
* Deploying and managing third-party virtual appliances in a scalable manner, such as security appliances.

**Example**: If you need to deploy a scalable security appliance to inspect and manage network traffic, GWLB can distribute traffic to multiple instances of the appliance.

**Summary**

1. **Classic Load Balancer (CLB)**
   * **Layer**: 4/7
   * **Use Case**: Simple applications needing basic HTTP/TCP load balancing.
2. **Application Load Balancer (ALB)**
   * **Layer**: 7
   * **Use Case**: Advanced HTTP/HTTPS applications with complex routing needs.
3. **Network Load Balancer (NLB)**
   * **Layer**: 4
   * **Use Case**: High-performance, low-latency TCP traffic.
4. **Gateway Load Balancer (GWLB)**
   * **Layer**: 3/4
   * **Use Case**: Scalable deployment of third-party network appliances.

Each type of load balancer is designed for specific needs, from basic load distribution to advanced routing and high-performance traffic management. Choosing the right one depends on your application’s requirements.



**Auto Scaling:**

Auto Scaling in AWS is a service designed to automatically adjust the number of Amazon EC2 instances (or other resources) in your application based on current demand. It helps ensure that your application has the right amount of capacity to handle incoming traffic while optimizing costs.

**Why Use Auto Scaling?**

1. **Cost Efficiency**: Automatically scale down resources during low traffic periods to save costs.
2. **High Availability**: Maintain application availability by automatically replacing unhealthy instances.
3. **Performance Optimization**: Scale up resources during high traffic periods to ensure your application performs well.
4. **Reduced Manual Intervention**: Automate capacity management to reduce the need for manual adjustments.

**When to Use Auto Scaling**

1. **Variable Traffic**: When your application experiences fluctuating traffic or workload.
2. **Growth**: When your application is growing and you need to ensure it can handle increased load.
3. **High Availability**: When you want to ensure your application remains available even if some instances fail.

**How to Use Auto Scaling**

**Example Scenario**

Let’s say you run a website with variable traffic throughout the day. During peak hours (e.g., 9 AM to 6 PM), you experience high traffic, and during off-peak hours, the traffic decreases. You want to ensure your website performs well during peak hours but doesn’t incur unnecessary costs during off-peak hours.

In AWS, **vertical auto scaling** and **horizontal auto scaling** are two different approaches to managing resource capacity.

**Vertical Auto Scaling**

**Definition**: Vertical auto scaling involves resizing the resources of an individual instance, typically by changing the instance type to one with more CPU, memory, or other resources. This is often referred to as "scaling up" or "scaling down."

**How It Works**:

* **Scaling Up**: Upgrading an instance to a larger size with more resources (CPU, memory, etc.) when current resources are insufficient.
* **Scaling Down**: Downgrading an instance to a smaller size with fewer resources when the current instance is over-provisioned.

**Example**: Imagine you run a database on an EC2 instance that occasionally experiences high memory usage.

1. **Initial Setup**: You start with an t3.medium instance that has 4 vCPUs and 16 GB of RAM.
2. **Scaling Up**: During periods of high load, the database performance suffers due to high memory usage. You decide to scale up to an r5.large instance with 8 vCPUs and 64 GB of RAM to handle the increased load.
3. **Scaling Down**: When traffic decreases and memory usage drops, you could scale down back to the t3.medium instance to save costs.

**Benefits**:

* **Simpler Management**: Only one instance to manage and configure.
* **Increased Performance**: Directly addresses performance issues by upgrading the instance’s resources.

**Limitations**:

* **Downtime**: Resizing an instance often requires stopping and restarting it, leading to potential downtime.

**Horizontal Auto Scaling**

**Definition**: Horizontal auto scaling involves adding or removing instances from a pool to handle changes in demand. This is often referred to as "scaling out" or "scaling in."

**How It Works**:

* **Scaling Out**: Adding more instances to distribute the load across a larger number of instances.
* **Scaling In**: Removing instances when the load decreases, to save costs and reduce resource usage.

**Example**: Consider a web application running behind a load balancer.

1. **Initial Setup**: You start with 3 EC2 instances handling web traffic.
2. **Scaling Out**: During a marketing campaign, traffic spikes significantly. To handle the increased load, you set up an Auto Scaling Group to automatically add 3 more instances, bringing the total to 6.
3. **Scaling In**: After the campaign ends and traffic drops, the Auto Scaling Group automatically terminates the 3 additional instances, reducing the total back to 3.

**Benefits**:

* **High Availability**: Distributes the load across multiple instances, reducing the risk of failure and improving reliability.
* **Flexibility**: Easily handles sudden spikes or drops in traffic by adjusting the number of instances.

**Limitations**:

* **Complexity**: Requires managing multiple instances and ensuring proper load balancing and session management.
* **Overhead**: Additional instances can lead to higher costs if not managed correctly.

**Summary of How It Works**

1. **Auto Scaling Group (ASG)**:
   * **Manages a collection of EC2 instances**.
   * **Maintains a desired number of instances** by automatically scaling in or out.
2. **Automatic Integration with Launch Templates**:
   * **Defines the configuration** of instances used by the ASG.
   * **Ensures consistency** in instance setup.
3. **Scaling Policies and Triggers**:
   * **Automates scaling actions** based on predefined conditions.
   * **Uses CloudWatch alarms** or scheduled actions to adjust the number of instances.
4. **Health Checks**:
   * **Monitors instance health** continuously.
   * **Replaces unhealthy instances** to maintain the desired capacity.

**Simple Example**

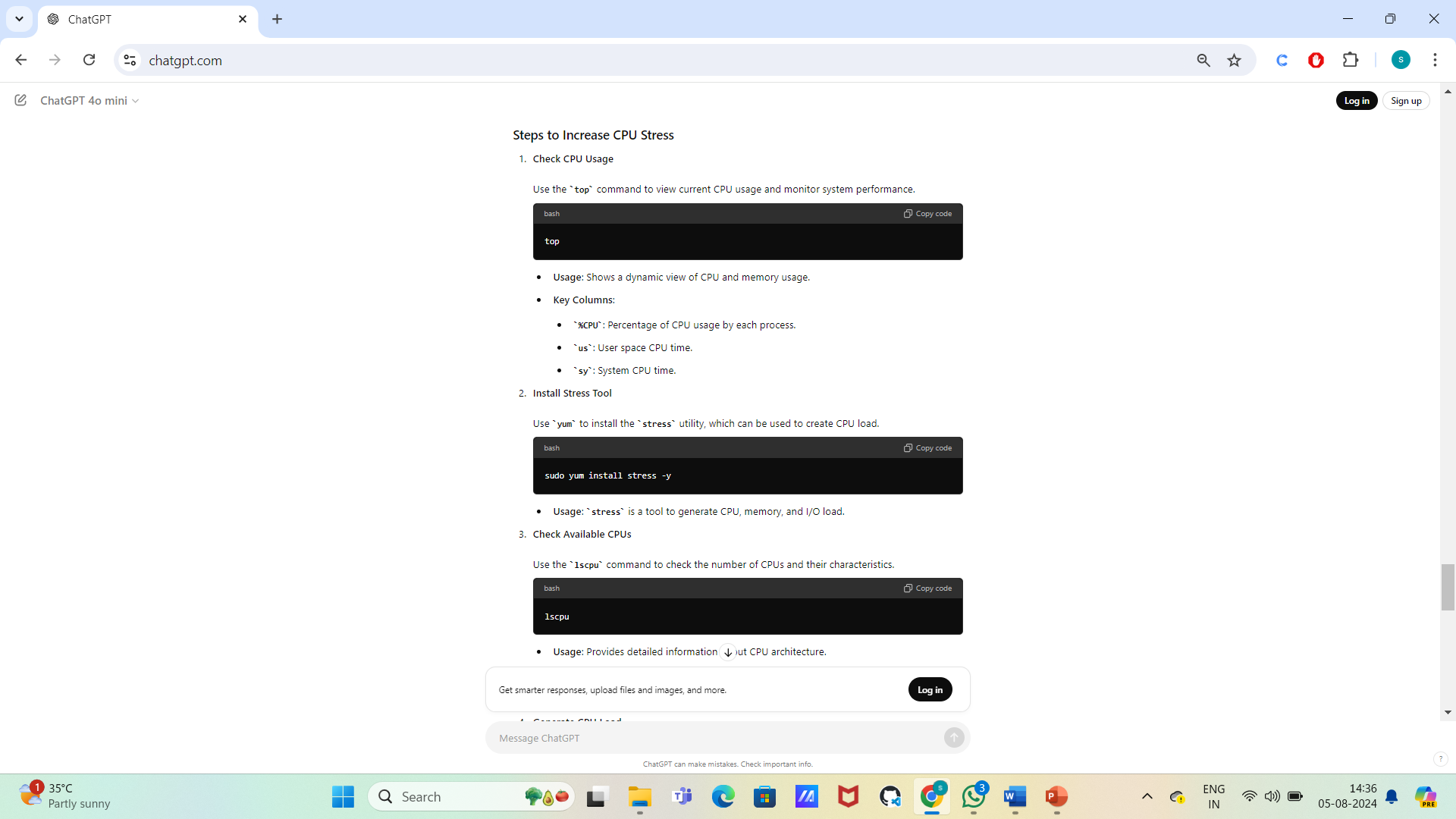
**Scenario**: You run a web application that experiences varying traffic levels throughout the day.

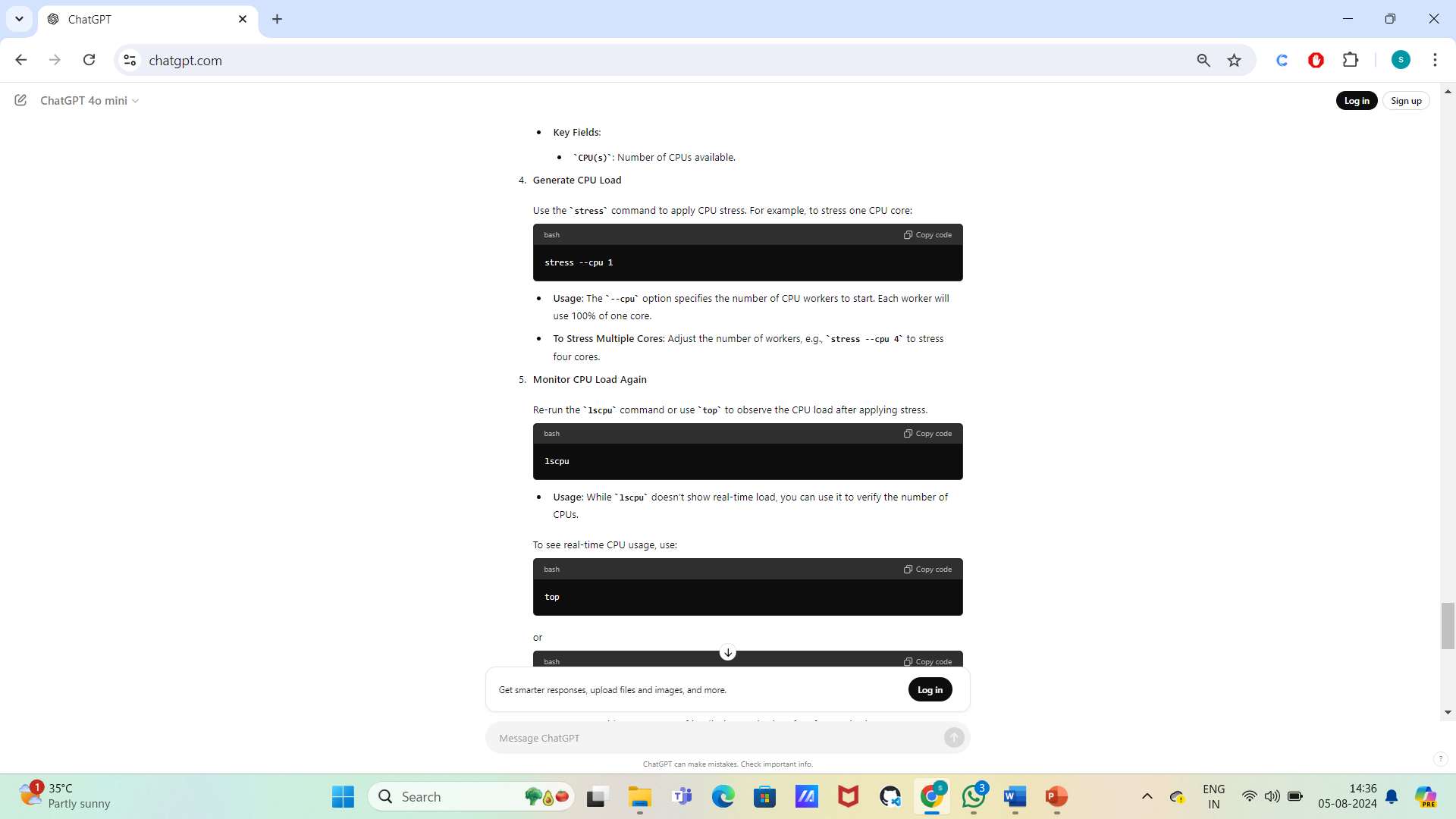
1. **Create a Launch Template**: Define the EC2 instance configuration (e.g., t3.medium).
2. **Set Up Auto Scaling Group**:
   * **Desired Capacity**: 2 instances.
   * **Minimum Instances**: 1 instance.
   * **Maximum Instances**: 5 instances.
   * **Attach Launch Template**.
3. **Define Scaling Policies**:
   * **Scale Out Policy**: Add 1 instance if CPU utilization exceeds 70% for 5 minutes.
   * **Scale In Policy**: Remove 1 instance if CPU utilization falls below 30% for 10 minutes.
4. **Configure Health Checks**:
   * **Health Check Type**: EC2.
   * **Action**: Replace any instance that fails health checks to maintain 2 healthy instances.

**Outcome**:

* During peak traffic, the ASG scales out by adding instances as needed.
* During low traffic, the ASG scales in by removing excess instances.
* If an instance becomes unhealthy, it is replaced automatically to ensure application stability.

This setup ensures your application remains performant and cost-effective by dynamically adjusting resources based on demand and maintaining instance health.





RDS:

 **Quick Setup and Deployment**:

* **Fast Initialization**: RDS allows you to set up a database instance quickly, typically within a few minutes. This rapid deployment helps accelerate development and testing processes.

 **Failover Capability**:

* **Automatic Failover**: With Multi-AZ deployments, RDS provides automatic failover to a standby instance in another Availability Zone (AZ) in case of a failure. This ensures higher availability and minimizes downtime.

 **Multi-AZ Deployments**:

* **Enhanced Availability**: Multi-AZ deployments replicate data across multiple availability zones. This architecture not only provides high availability but also improves fault tolerance. In the event of an AZ failure, RDS automatically promotes the standby instance to become the primary database.

 **Automated Backups**:

* **Regular Backups**: RDS performs automated backups of your database daily, and you can retain these backups for up to 35 days. This includes snapshots of the database and transaction logs, enabling point-in-time recovery.
* **Easy Restoration**: You can restore your database to any specific point within the backup retention period, providing flexibility in data recovery.

 **Scalability**:

* **Dynamic Scaling**: You can easily scale your RDS instance's compute and storage resources up or down based on your needs. This allows you to handle varying workloads efficiently without manual intervention.

 **Maintenance and Patching**:

* **Managed Patching**: RDS takes care of routine maintenance tasks such as applying patches and updates to the database engine, reducing the administrative burden on your team.

Amazon RDS (Relational Database Service) is a fully-managed service for setting up, operating, and scaling relational databases in the cloud. Here are two key points:

1. **Automated Database Management**: RDS takes care of common database administration tasks like backups, patch management, monitoring, and scaling, allowing you to focus on your application rather than managing the database infrastructure.
2. **Supports Multiple Database Engines**: RDS supports various database engines including MySQL, PostgreSQL, MariaDB, Oracle, SQL Server, and Amazon Aurora. This allows flexibility in choosing the best database system for your workload needs.

practical explanation of Amazon RDS:

1. **Creating a Database Instance**:
   * In AWS Management Console, navigate to RDS, select "Create Database," choose your database engine (e.g., MySQL), specify instance size, storage type, and security settings.
   * AWS will handle the provisioning of the database, backups, and scaling resources as needed.
2. **Connecting to the RDS Database**:
   * After RDS is set up, you can connect to the database from your application by using the endpoint URL provided by AWS RDS.
   * For example, if you're using MySQL, you would connect via MySQL Workbench or an application using connection parameters like host (endpoint), port, database name, and credentials.

Example MySQL connection string:

mysql -h your-db-endpoint.rds.amazonaws.com -u admin -p

In practice, this simplifies the management of relational databases, as AWS takes care of patching, backups, and scaling, while you focus on developing your application.

4o