set PATH=%PATH%;%USERPROFILE%\awscli\Amazon\AWSCLIV2

**Set the path for AWS CLI**

1. **What is virtualization?**

**Virtualization** is the process of creating a **virtual (software-based) version of hardware resources**, as:-

* **Virtual machines (VMs)** (virtual computers)
* **Virtual storage (e.g., virtual disks)**
* **Virtual networks**
* **Virtual desktops or applications**

It allows **multiple operating systems or applications** to run on the **same physical hardware** by **abstracting hardware resources** through a **virtualization layer (hypervisor)**.

**Hypervisor:** A **Hypervisor** is **software** that allows you to **create and manage virtual machines (VMs)** on a **single physical machine** by splitting and managing hardware resources like CPU, RAM, and storage.

**Types of Hypervisors**:

|  |  |  |
| --- | --- | --- |
| **Type** | **Description** | **Examples** |
| **Type 1 (Bare Metal)** | Runs **directly on hardware**, best for servers. | VMware ESXi, Microsoft Hyper-V, KVM, Xen |
| **Type 2 (Hosted)** | Runs **on top of a host OS**, good for desktops/laptops. | VirtualBox, VMware Workstation, Parallels Desktop |

|  |  |
| --- | --- |
| [Hardware]  ↓  [Host OS (Windows/Linux/macOS)]  ↓  [Hypervisor **(Type 2)]**  ↓  [VM1] [VM2] | [Hardware]  ↓  [Hypervisor **(Type 1)]**  ↓  [VM1] [VM2] [VM3] |

|  |  |  |
| --- | --- | --- |
| **Feature** | **Type 1 Hypervisor** | **Type 2 Hypervisor** |
| **Installation** | Directly on hardware | On top of the existing OS |
| **Performance** | High (close to hardware) | Lower (overhead from host OS) |
| **Use Case** | Servers, data centers | Personal laptops, testing |
| **Examples** | VMware ESXi, KVM, Hyper-V | VirtualBox, VMware Workstation |
| **Security** | More secure | Less secure (depends on host OS) |
| **Ease of Use** | Needs expertise | Easier, beginner-friendly |
| **Cost** | May require licenses | Often free or cheaper |

✅ **Hardware Efficiency** → Run multiple servers on one machine.

✅ **Isolation** → Crash in one VM doesn’t affect others.

✅ **Flexibility** → easily creates, deletes, and clones VMs.

✅ **Cost Savings** → Fewer physical machines are needed.

1. **What is cloud computing?**

**Cloud computing** means **accessing IT resources like servers, storage, databases, networking, and software via the internet** instead of buying or maintaining physical hardware.

**Cloud Computing Service Models:**

|  |  |  |
| --- | --- | --- |
| **Model** | **Meaning** | **Example** |
| **IaaS** (Infrastructure as a Service) | Rent virtual machines, storage | AWS EC2, Azure VM, Google Compute Engine |
| **PaaS** (Platform as a Service) | Get ready-to-use platforms to deploy apps | Heroku, Google App Engine, AWS Elastic Beanstalk |
| **SaaS** (Software as a Service) | Use ready-made software over internet | Gmail, Google Drive, Microsoft 365 |

**IaaS: You manage**: OS + Applications. **Cloud manages**: Physical Server + Virtualization

**Paas**: **focus only on code**, no need to manage OS, middleware, or runtime.

**You manage**: Code + Configuration. **Cloud manages**: Infrastructure + OS + Middleware + runtime

**Deployment Models (Where Cloud is Hosted):**

|  |  |  |
| --- | --- | --- |
| **Type** | **Description** | **Example Use-Case** |
| **Public Cloud** | Services are offered over the public internet and shared across multiple organizations. | AWS, Azure, GCP |
| **Private Cloud** | Services are maintained on a private network for a single organization (more secure). | On-premises cloud (VMware), OpenStack |
| **Hybrid Cloud** | Combination of public and private clouds, with data and apps shared between them. | Sensitive data on private cloud, other apps on public cloud |
| **Community Cloud** | Shared infrastructure for a specific community with common concerns (security, compliance). | Government departments, healthcare groups |

**Common Features of Cloud Providers (AWS, Azure, GCP, etc.):**

|  |  |
| --- | --- |
| **Feature** | **Description** |
| **On-Demand Self-Service** | You can provision resources anytime without human interaction (via portal or API). |
| **Broad Network Access** | Services are accessible over the internet from anywhere via standard devices (laptop, mobile). |
| **Rapid Elasticity** | Easy to scale resources up or down automatically based on demand (autoscaling). |
| **Disaster Recovery** | Built-in backup, snapshots, and recovery options across regions. |
| **Security & Compliance** | Built-in security features (encryption, IAM, firewalls) and compliance with standards like GDPR, ISO, PCI-DSS. |
| **Global Reach** | Data centers (regions/zones) around the world, allowing low latency and regional redundancy. |
| **Automation & APIs** | Everything is programmable using APIs, SDKs, and infrastructure as code (Terraform, CloudFormation). |
| **Monitoring & Logging** | Tools for monitoring (CloudWatch, Azure Monitor), logging, and alerting. |
| **Managed Services** | Pre-configured services for databases (RDS), AI/ML, analytics (BigQuery), etc., reducing management overhead. |

1. **What is AWS**

It offers **on-demand cloud services** like **computing power, storage, databases, machine learning, and more**, all accessible via the internet on a **pay-as-you-go** basis.

**Popular AWS Services:**

|  |  |  |
| --- | --- | --- |
| **Category** | **Examples** | **Purpose** |
| **Compute** | EC2, Lambda | Virtual servers (EC2), serverless computing (Lambda) |
| **Storage** | S3, EBS, Glacier | Object storage (S3), block storage (EBS), archival backups (Glacier) |
| **Content Delivery** | CloudFront | delivers data, videos, applications, and APIs to users globally with low latency and high transfer speeds. |
| **Databases** | RDS, DynamoDB, Aurora | Managed SQL (RDS, Aurora) and NoSQL (DynamoDB) databases |
| **Networking** | VPC, Route 53, Elastic Load Balancer | Private networking (VPC), DNS (Route 53), load balancing (ELB) |
| **Security** | IAM, KMS, AWS Shield | Identity and access management (IAM), encryption (KMS), DDoS protection (Shield) |
| **AI/ML** | SageMaker, Rekognition | Machine learning (SageMaker), image and video analysis (Rekognition) |
| **Monitoring** | CloudWatch, CloudTrail | Resource monitoring (CloudWatch), API activity logging (CloudTrail) |

**AWS Account**: vishalk00001497@gmail.com/Sheetal3021@

**AWS is special because of**: Scalability, Global Reach, Reliability, Security

1. **IAM (Identity and Access Management):**

**AWS IAM** is a **secure access management service** in AWS that **controls who can access AWS resources** and **what actions they can perform**.

IAM allows you to:

* **Create users and groups**
* **Assign specific permissions**
* **Manage secure access** to AWS services

**Features of IAM:**

* **Fine-grained Access Control** → Allow/deny actions on specific AWS resources
* **Multi-Factor Authentication (MFA)** → Add extra security layer
* **It is a global-level service**
* **Temporary Credentials** → Roles for EC2, Lambda, or external users
* **Federated Access** → Login with corporate directory (**Active Directory)** or third-party identity (Google, Okta)
* **Free Service** → IAM itself is free; you only pay for the AWS resources accessed

**Tags While user creation:** Tags are key-value pairs you can add to AWS resources to help **identify**, **organize**, or **search** for resources.**Ex. Department, Role, Environment, etc**

**How to force users to set up MFA:**

You can create a policy like this to deny all actions if MFA is not enabled:

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

"Statement": [{

"Sid": "**DenyAllExceptMFA**",

"Effect": "Deny",

"Action": "\*",

"Resource": "\*",

"Condition": {

"BoolIfExists": {

"aws:MultiFactorAuthPresent": "false"

}}}]}

**Ways to Access AWS Resources:**

**1. AWS Management Console(Web UI):** Web-based graphical interface to manage AWS.

**2. AWS CLI (Command Line Interface):** Automate tasks using bash or scripts.

**3. AWS SDKs (Software Development Kits):** Programmatic access to AWS services by code.

**4.** **AWS CloudShell**: Browser-based terminal pre-installed with AWS CLI, Python, Git, etc.

**AWS common CLI Command:**

|  |  |
| --- | --- |
| **Command** | **Use Case** |
| **aws configure** | Configure AWS credentials, region, and output format. |
| **aws s3 ls** | List all S3 buckets. |
| **aws s3 ls s3://my-bucket/** | List files inside a specific S3 bucket. |
| **aws s3 cp file.txt s3://my-bucket/** | Upload file to S3 bucket. |
| **aws s3 cp s3://my-bucket/file.txt .** | Download file from S3 bucket. |
| **aws s3 sync ./localdir s3://my-bucket/** | Sync local directory to S3 bucket (good for backups). |
| **aws ec2 describe-instances --output table** | List EC2 instances in table format. |
| **aws ec2 start-instances --instance-ids i-xxxxxxxx** | Start a stopped EC2 instance. |
| **aws ec2 stop-instances --instance-ids i-xxxxxxxx** | Stop a running EC2 instance. |
| **aws ec2 reboot-instances --instance-ids i-xxxxxxxx** | Reboot an EC2 instance. |
| **aws iam list-users** | List all IAM users. |
| **aws iam get-user** | Get details of the current IAM user. |
| **aws sts get-caller-identity** | Check the current IAM identity and AWS account. |
| **aws logs describe-log-groups** | List CloudWatch log groups. |
| **aws logs describe-log-streams --log-group-name my-log-group** | List log streams in a log group. |
| **aws cloudwatch list-metrics** | List available CloudWatch metrics. |
| **aws cloudformation list-stacks** | List CloudFormation stacks. |
| **aws cloudformation describe-stacks** | Get detailed stack information. |
| **`aws ecr get-login-password** | docker login --username AWS --password-stdin .dkr.ecr..amazonaws.com` |
| **aws lambda list-functions** | List deployed Lambda functions. |

**Best Practices for AWS IAM**:

|  |  |
| --- | --- |
| **Best Practice** | **Purpose / Benefit** |
| **Enable MFA for all users** | Protect accounts from credential theft. |
| **Never use the root account except for setup** | Prevent accidental damage or breaches. |
| **Apply the Least Privilege Principle** | Only allow necessary permissions to reduce the attack surface. |
| **Use IAM Roles (not Users) for EC2/Lambda** | Avoid embedding long-term credentials in code. |
| **Rotate Access Keys Regularly** | Limit risk from compromised keys. |
| **Use IAM Groups for Permission Management** | Simplify and standardize permission assignments. |
| **Enable CloudTrail for Auditing** | Track all account-level activities for security audits. |
| **Use IAM Access Analyzer** | Detect unintended resource sharing. |
| **Use Policy Conditions (IP, MFA, Time)** | Add context-aware access controls. |
| **Review and Remove Unused Permissions Regularly** | Clean up stale access to minimize risks. |

**Password Policy:**

* 1. Go to **AWS Console → IAM → Account Settings**.
  2. Using AWS CLI: **aws iam update-account-password-policy \**

**--minimum-password-length 12 \**

1. **What is EC2 (Elastic Computing)?**

**Amazon EC2 (Elastic Compute Cloud)** is a **virtual server in the cloud** provided by AWS. It allows you to run applications on **virtual machines** (VMs) with complete control over computing resources.

Think of it as a **rental computer** in the cloud, which you can scale up or down as needed.

**Why use?**

|  |  |
| --- | --- |
| **Benefit** | **Explanation** |
| **Scalable** | Easily scale up (more CPU/RAM) or down as per need. |
| **Flexible** | Full control over OS, software, and configurations. |
| **Pay-as-you-go** | You only pay for what you use (per second/minute/hour). |
| **Global** | Can launch servers globally across AWS Regions. |
| **Secure** | Integrates with IAM, Security Groups, key pairs for secure access. |

**Common EC2 Use Cases**

* Hosting websites and APIs (Spring Boot, Liferay, Node.js)
* Running backend microservices
* Batch processing jobs
* Temporary compute capacity for development/testing
* Hosting Docker containers or Kubernetes clusters (EKS)

**EC2 Basic Component:**

|  |  |
| --- | --- |
| **Component** | **Purpose** |
| **Instance** | A virtual server running your application. |
| **AMI (Amazon Machine Image)** | choose OS + software. |
| **Instance Type** | Defines CPU, RAM, storage (e.g., t3.micro, m5.large). |
| **Key Pair** | Create or use SSH credentials to connect to your instance. |
| **Security Group** | Set up a firewall rule controlling inbound/outbound traffic. |
| **Elastic IP** | A static IP address you can attach to an instance. |
| **Storage** | Configure the type and size of storage. Ex. **EBS Volume** |
| **IAM Role** | Attach an IAM role for permission to access Other AWS resources |
| **Network Setting** | Configure VPC, subnet, and assign a public or private IP |
| **User Data** | Add scripts to be executed when the instance starts |

* EC2 services are region-specific.
* Create an EC2
* We can provide a Script to run while the instance starts, within **advanced detail tab.**

#!/bin/bash

sudo yum update -y

**# Install Apache web server (httpd)**

sudo yum install -y httpd

sudo systemctl start httpd

sudo systemctl enable httpd

**# Create a simple HTML file to verify the web server is running**

echo "<html><h1>Welcome to Apache Web Server on Amazon Linux!</h1></html>" > /var/www/html/index.html

**Security Group:** In the security group, We can add a rule for inbound and outbound tariff and whitelist the IP. Ex. For HTTP port 80 & SSH port 22

**Inbound Rules:** To allow any new IP, need to add a new rule from security group.

We can add multiple security groups to an EC2 instance.

|  |  |
| --- | --- |
| **What It Controls** | **Explanation** |
| **Incoming traffic TO your EC2 instance** | Who can **access** your EC2 (from the outside). |
| **Use Case** | Allow specific IPs to SSH (port 22), HTTP (port 80), or HTTPS (port 443) into the server. |
| **Example Rule** | Allow only your office IP 203.0.113.5/32 to connect via SSH: Port 22 |
| **Important Rule** | If no **Inbound Rule**, nobody can access your server from outside. |

**Outbound Rules:**

|  |  |
| --- | --- |
| **What It Controls** | **Explanation** |
| **Outgoing traffic FROM your EC2 instance** | Where your EC2 can **connect to** (internet or other AWS resources), .ex, add Apache using the internet |
| **Use Case** | Allow EC2 to download updates, connect to S3, or external APIs. |
| **Default Rule** | By default, **all outbound traffic is allowed** (open 0.0.0.0/0). |
| **Example Rule** | Allow only connections to a private database in VPC or restrict internet access for security reasons. |

**Important Points About Security Groups**

|  |  |
| --- | --- |
| **Key Point** | **Explanation** |
| **Scope** | Region Specific. For other regions, reuse the same group for all regions by manually adding or automating using CLI |
| **Stateful** | Responses to **allowed inbound traffic** are automatically **allowed outbound** (and vice versa). No need to define both directions. |
| **Attached to Network Interface** | Security Groups are attached to **ENI (Elastic Network Interface)**, not directly to the EC2 instance. |
| **Only Allow Rules (No Deny)** | You can **allow traffic**, but **cannot create deny rules** (denials happen by omission). |
| **Default Behavior** | By default: **Inbound = Denied**, **Outbound = Allowed**. |
| **Can be Assigned to Multiple Instances** | One Security Group can be attached to **many EC2 instances**, making management easier. |
| **Rules by IP or Security Group** | Allow traffic by **CIDR blocks (IP addresses)** or by **other Security Groups** (good for internal traffic). |

**WAF on EC2:** AWS WAF **cannot be applied directly to EC2 instance IP** — it filters traffic **before it reaches EC2**, via **ALB(Application Load Balancer) or API Gateway,** or **CloudFront**.

**Connect EC2 by SSH steps:**

1. **.**pem file is downloaded while instance->key pair creation.
2. Load this file in PuTTYgen. And save the private key as extension .key
3. Open Putty and load the .key file within SSH -> Auth ->credentials
4. In Linux just locate in .pem file and hit a command mention in dashboard

**Instance Type use case:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case** | **Recommended Instance Type** | | **Category** |
| **Small Website or Blog** | | | t3.micro or t3.small | General Purpose |
| **E-Commerce Application** | | | m5.large or m5.xlarge | General Purpose |
| **Real-Time Video Rendering and Streaming** | | | g5.12xlarge or g5.24xlarge | Accelerated Computing |
| **In-Memory Database for Real-Time Analytics** | | | r6g.16xlarge or x2idn.32xlarge | Memory Optimized |

1. **What is AWS EBS?**

**Amazon Elastic Block Store (EBS)** provides **block-level storage volumes** for **EC2 instances**.  
Think of it as a **hard drive attached to your EC2**, allowing persistent, durable storage that remains even after an instance stops or terminates (unless deleted).

It is region & zone-specific. If we terminate(delete) the EC2 volume, the default root volume will also be deleted. Because It is related to OS.

But we can edit the setting by setting “delete on termination ” to false in the EC2 instance.

We can create and attach multiple volumes to an EC2 instance. Just create and attach a volume to the EC2 instance.

**Key Features of EBS**:

|  |  |
| --- | --- |
| **Feature** | **Description** |
| **Durable** | Data is replicated within the same Availability Zone (AZ). |
| **Persistent Storage** | Data remains after EC2 stops/reboots. |
| **Scalable** | Resize volumes anytime (expand without downtime). |
| **Backup (Snapshots)** | Easy to back up using EBS Snapshots (stored in S3). |
| **Encryption** | Supports encryption at rest and in transit. |
| **High Availability** | Replicated across multiple physical devices in an AZ. |

**EBS Volume Types**

|  |  |  |
| --- | --- | --- |
| **Type** | **Use Case** | **Performance** |
| **gp3 (General Purpose SSD)** | Balanced workloads, default choice | Up to 16,000 IOPS |
| **gp2 (Legacy General Purpose SSD)** | Older workloads | Performance scales with size |
| **io2/io2 Block Express (Provisioned IOPS SSD)** | Databases, high-performance apps | Up to 256,000 IOPS (io2 Block Express) |
| **st1 (Throughput Optimized HDD)** | Big data, log processing | High throughput, lower IOPS |
| **sc1 (Cold HDD)** | Archival, infrequently accessed | Lowest cost, lowest performance |

**Basic EBS Concepts**

|  |  |
| --- | --- |
| **Concept** | **Meaning** |
| **Volume** | The storage disk you attach to EC2. |
| **Snapshot** | Backup of your EBS volume to S3 (incremental). |
| **Attachment** | Can attach/detach to/from running EC2 instances (except root volumes). |
| **Root Volume** | EBS volume from which the EC2 OS boots. It holds the OS. Create **automatically while launching EC2.** |
| **Non-root Volume** | Additional attached storage (e.g., /dev/xvdf). |

**EBS Snapshot:** take a snapshot from ABS volume. Now using this snapshot we can **create a volume in different Zone**. Using this volume We can create a EC2 instance.

**Note**: We can create a volume in different region using copy a snapshot(change its region here)

While creating Ec2 instance We can choose region within **networking->subnet**

In this way can create EC2 instance using different region storage data.

**How to mount Storage disk:**

If one EC2 instance has multiple ABS/disk. So We can use other using mount it in a folder.

* Lsblk //Verify attached disk
* Step: **sudo file -s** /dev/nvmelnlpl //for attaching/format file system
* Sudo mkdir /foldername //for creating mount directory
* Sudo mount -o nouuid /dev/nvmelnlpl /foldername //for mounting
* Sudo unmount -l /foldername //for unmount directory

1. **Amazon Data Lifecycle Manager:**

**Amazon Data Lifecycle Manager (DLM)** is an AWS service that helps you **automatically manage EBS snapshots** and **EBS-backed AMIs** on a schedule. Mostly, we use it for taking a backup policy.

**Why Use It?**

* **Automation** → no manual snapshot/AMI management.
* **Cost Saving** → auto-deletes old snapshots/AMIs.
* **Compliance** → keeps backups for defined retention periods.

**Common Use Cases**:

|  |  |
| --- | --- |
| **Use Case** | **How DLM Helps** |
| **Daily backups** | Automatically take daily EBS snapshots. |
| **Retention policy** | Keep the last 7 days of snapshots, delete older ones. |
| **Golden AMIs** | Automatically create AMIs weekly/monthly for EC2 scaling. |

**AMI (Amazon Machine Image):**

**An Amazon Machine Image (AMI) is a pre-configured template that provides the necessary information to launch an EC2 instance in AWS.**

**UAT to PROD with same configuration**: Just create an Image of the existing instance, then create an EC2 instance using this image.

If we create a PROD environment using the UAT Image. Then we need to remove multiple things like as Testing users, unwanted permissions, etc.

It includes:

* Operating system (e.g., Linux, Windows)
* Application server (e.g., Apache, Nginx)
* Pre-installed software and configurations

|  |  |
| --- | --- |
| * #!/bin/bash * # Clear user credentials and history * rm -rf ~/.aws/credentials * rm -rf ~/.git-credentials * rm -rf ~/.bash\_history * # Clean system logs and temporary files * rm -rf /var/log/\* * rm -rf /tmp/\* * rm -rf /var/tmp/\* | * # Remove user accounts * deluser tempuser --remove-home * # Lock root account * passwd -l root * # Reset configuration files (example for Nginx) * rm -rf /etc/nginx/nginx.conf * # Remove SSH keys * rm -rf ~/.ssh/authorized\_keys |

**Types of AWS AMIs**

* **Public AMIs:** Available to all AWS users. Useful for basic use cases like popular operating systems (e.g., Ubuntu, CentOS).
* **Private AMIs:** Created by a user and only available within that account or shared with specific accounts.
* **Paid AMIs/Marketplace AMIs:** Provided by third parties through AWS Marketplace, offering software like databases, web servers, or pre-configured environments.

|  |  |
| --- | --- |
| **Image** | **Template** |
| A **copy of a machine** with everything installed (OS, software, settings). You can **quickly launch** a server from it. | A **blueprint or instruction file** that tells AWS or any system **what to create** (servers, databases, networks).  Ex. Security Group, OS Type, Network setting, Storage |
| Like a **photo** of a fully set-up computer. | Like a **recipe** for building a computer from scratch. |

**What is EC2 Image Builder?**

**EC2 Image Builder** is an AWS service that helps you **automate the creation, testing, and distribution of EC2 machine images (AMIs)**.

**Why Use EC2 Image Builder?**

* **Automatically build AMIs** on a schedule (daily, weekly, monthly).
* **Add security patches automatically** (OS updates, antivirus).
* **Pre-install your applications** and configurations.
* **Test images** before use (to avoid broken servers).
* **Cost-effective** → fully managed, no extra charge (you pay only for resources used).

**ELB(Elastic Load Balancer):**

**ELB (Elastic Load Balancer)** is an AWS service that **distributes incoming traffic** across multiple targets like **EC2 instances, containers, or IP addresses**, making your application **highly available** and **scalable**.

Instances can be in any zone.

In AWS, while creating a **Load Balancer (LB)**, you **don’t directly edit DNS** during creation — instead: **After creation**, you can **map your custom domain** (e.g., www.example.com) to the Load Balancer DNS using **Route 53** or any DNS provider.

**Types of ELB**:

|  |  |
| --- | --- |
| **Type** | **Use Case** |
| **Application Load Balancer (ALB)** | Best for **HTTP/HTTPS**, advanced routing, microservices, and path-based routing. |
| **Network Load Balancer (NLB)** | Best for **TCP/UDP**, high performance, millions of requests per second, very low latency. |
| **Gateway Load Balancer (GWLB)** | Used for **third-party appliances** like firewalls. |
| **Classic Load Balancer (CLB)** | Legacy option, simple load balancing (not recommended for new apps). |

**Steps for Load Balancer:**

* Click on Create load balancer.
* Select load balancer type (App Load Balancer).
* Configure Load Balancer (name, Internet facing, subnets, Enable **listeners** (usually **HTTP:80** or **HTTPS:443**).
* Configure Security Group (Allow traffic on **listener ports** (e.g., 80 or 443).
* Configure Target Group: Select EC2 instances → **Add and register to target group**.
* Test Load Balancer
* On your **Web Server instances**, you need to configure **Reverse Proxy** to forward requests from the Web server to the app(**Liferay) instances**.

**After Creation: You** will get a **DNS Name** → Use this in your browser or application.

Traffic will be **balanced across healthy EC2 instances**.

While creating a Load Balancer, we can also apply WAF.

**Scalability**: Scalability means the ability to grow your system’s resources when your application or website gets more traffic.

**Vertical Scalability (Scaling Up)**

* Vertical Scalability means adding more power (CPU, RAM) to your existing server.
* Ex: t2.micro to m5.large

**Horizontal Scalability (Scaling Out)**

* Horizontal Scalability means adding more instances (servers) to distribute the load.
* You can add more EC2 instances behind a load balancer.

**High Availability (HA):** High Availability means keeping your service up and running with minimal downtime, so it's always accessible to users.

* **Ex:** running resources in multiple AZs

**Elasticity: Elasticity** means the ability to automatically adjust resources as the demand changes—adding more when needed and removing when it’s no longer necessary.

* **Ex:** **ASG(Auto Scaling Group)**

**What is ASG (Auto Scaling Group) in AWS?**

**ASG (Auto Scaling Group)** is an AWS feature that automatically **launches or terminates EC2 instances** based on demand, ensuring your application always has the **right amount of capacity**.

**Why Use ASG?**

|  |  |
| --- | --- |
| **Benefit** | **Explanation** |
| **Automatic Scaling** | Adds or removes EC2 instances automatically. |
| **Elasticity** | Handles traffic spikes without manual intervention. |
| **High Availability** | Replaces unhealthy instances automatically. |
| **Cost Optimization** | Scales in when traffic is low, reducing costs. |

**Features:**

**1. Automatic Scaling:** Scale the number of EC2 instances up or down based on demand.

**2. Maintain Instance Health:** Replace unhealthy instances automatically to ensure reliability.

**3. Use Scaling Policies:** Set rules for scaling based on metrics like CPU usage or request count.

**4. Ensure Availability:** Always keep a defined number of instances running to meet app needs.

**5. Schedule Scaling:** Pre-configure scaling activities for specific times (e.g., traffic peaks).

**6. Distribute Instances**: Deploy instances across multiple Zones to ensure high availability.

**7. Integrate with ELB**: Use a Load Balancer to automatically distribute traffic across instances.

**8. Optimize Costs**: Scale down during periods of low demand to reduce infrastructure expenses.

**Steps to create ASG:**

• Create an AMI, using this AMI, **create/launch** a template with a security group.

• **Launch Template/image or Configuration** – Define the instance settings like AMI, instance type, key pair, etc.

• **Create Auto Scaling Group** – create using a template, set up the group that will manage instance scaling.

• **Select VPC and Subnets** – Choose the networking environment for the group.

• **Attach Load Balancer (Optional)** – Optionally link an ELB to distribute traffic.

• **Configure Scaling Policies** – Set rules to scale in or out based on demand. Ex. CPU utilization on request count.

• **Health Checks** – Enable checks to ensure instances are functioning properly.

• **Add Notifications (Optional)** – Configure alerts for scaling events.

• **Review and Create** – Final review before deploying the Auto Scaling Group.

**------------------------------------ S3 Bucket --------------------------------------------**

* **S3 Versioning**
* **S3 Replication**
* **Data Encryption**
* **S3 Bucket Policies**
* **S3 Storage Classes**
* **Logging Monitoring**
* **Hosting a Static Website**
* **Snow Family**
* **Storage Gateway (Hybrid Solution)**

**Amazon S3 (Simple Storage Service)** is an object storage service that allows you to store and retrieve **any amount of data** from **anywhere** on the internet.

An **S3 Bucket** is like a **folder** in the cloud where you can store files (called **objects**) such as:

* Documents, images, videos, backups, static websites, log files, etc

The bucket name should be unique in the global. **Region specific but data will replicate in all zone of that region.**

|  |  |  |
| --- | --- | --- |
| **Feature** | **S3 Bucket (Object Storage)** | **EBS Volume (Block Storage)** |
| **Use Case** | Storing static files, backups, logs, media, and static websites | Attaching storage to EC2 for running OS, databases, and applications |
| **Attach to EC2?** | ❌ Cannot attach directly | ✅ Directly attached to EC2 instances |
| **Access Method** | Access via **HTTP/HTTPS**, SDK, AWS Console, or CLI | Mount as a **filesystem** (like /dev/xvdf) on EC2 |
| **Persistence after EC2 termination** | ✅ Always persists (even without EC2) | ✅ If volume is not deleted, persists; else may be terminated with EC2 |
| **Scalability** | Unlimited storage (you don’t worry about size) | Fixed size (e.g., 100GB), can be resized, but with limits |
| **Pricing** | Pay-per-GB stored + requests (very cheap for backups) | Pay-per-GB provisioned (even if idle), more expensive for storage |

**Bucket Versioning: S3 Bucket Versioning** is a feature that **keeps multiple versions of an object (file)** in a bucket.  
Even if you **modify** or **delete** a file, **older versions are still stored safely**.

**Object Lock:** Amazon **S3 Object Lock** prevents objects (files) from being **deleted** or **overwritten** for a set period or indefinitely.  
It is mainly used to meet **compliance**, **legal hold**, or **data protection** needs.

**Why Use Object Lock?**

|  |  |
| --- | --- |
| **Use Case** | **Example** |
| **Compliance** | Keep files undeletable for 7 years to follow legal regulations. |
| **Ransomware Protection** | Prevent accidental or malicious deletion of critical backups. |
| **Data Retention** | Make sure logs, backups, and reports are retained safely. |

**What is Static Website Hosting in S3?**

**Amazon S3 Static Website Hosting** lets you **host a simple website** (HTML, CSS, JS files) **directly from S3**, **without any server** (no EC2 or backend).

**How to Set Up (Steps):**

**1. Create an S3 Bucket:** Name: same as your domain (e.g., example.com).

**2. Upload Your Website Files:** Example: index.html, about.html, css/style.css, etc.

**3. Enable Static Website Hosting**

* Go to **Bucket → Properties → Static website hosting → Enable**
* Set: Index document = index.html, & error.html(optional)

**4. Make Files Public**

* Go to **Permissions → Bucket Policy → Add: permissions**

**5. Access via S3 Website URL**

**S3 Bucket Policies**

JSON-based access control policies that you attach directly to an S3 bucket **to manage permissions for accessing the bucket and its objects.**

They allow you to define who can access the data and **what actions they can perform**, such as **read, write, or delete**, enabling fine-grained control over the security of your data stored in S3.

• **Write or paste your JSON policy in the Bucket Policy editor.**

• You can use **AWS’s Policy Generator** to create a custom policy, or you can manually write the policy in JSON format.

* **GetObject**: Used to retrieve or download files from an S3 bucket.
* **PutObject**: Used to upload or add files into an S3 bucket.

**S3 Replication**

It allows you to automatically copy objects from one S3 bucket to another, which can be

* within the same region (**Same-Region Replication - SRR**) or
* in different regions (**Cross-Region Replication - CRR**).

It’s commonly used for **compliance, redundancy, and to improve data access performance** by maintaining copies closer to your users.

**Steps**: Go to your S3 bucket->management->add replication rule-> and provide source & destination address, etc., for this **versioning must be enabled.**

**S3 Storage classes:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S3 Storage Class** | **Use Case** | **Features** | **Cost** |
| **S3 Standard** | Frequently accessed data | High durability, high availability, low latency | Most expensive, designed for frequent access |
| **S3 Intelligent-Tiering** | Data with unknown or unpredictable access patterns | Moves data automatically b/w frequent & infrequent tiers | Slightly higher than Standard but cost-saving based on usage |
| **S3 Standard-IA** | Infrequently accessed but quickly retrievable | Lower cost for storage, higher cost for data retrieval | Lower than Standard, ideal for less frequent access |
| **S3 One Zone-IA** | Non-critical, infrequently accessed data | Stored in a single Availability Zone, lower resilience | Cheaper than Standard-IA, good for non-critical data |
| **S3 Glacier** | Archival data, rarely accessed | Very low storage cost, retrieval time from minutes to hours | Ideal for long-term archives with low cost |
| **S3 Glacier Deep Archive** | Deep archival data, almost never accessed | Lowest cost, retrieval time up to 12 hours | **Cheapest storage, ideal for compliance or long-term retention** |
| **S3 Outposts** | Local storage using AWS Outposts | Provides S3 API locally, meets on-premises requirements | Dependent on Outposts infrastructure usage |

**S3 Bucket Lifecycle**

You can use lifecycle policies to control the **movement of objects** between different storage classes or delete them entirely, based on specific **conditions like age or inactivity.**

Create a **Lifecycle rule** for this for all or some objects. **Ex: prefix**

**The S3 Snow Family**

The S3 Snow Family is a group of physical devices offered by AWS to help move large amounts of data to the cloud when using the internet isn’t practical.

These devices are used when there's too much data to upload over a regular connection or when dealing with remote areas without good internet.

**The Snow Family includes:**

* **AWS Snowcone**: A small, portable device for a few terabytes of data.
* **AWS Snowball**: A larger device for moving petabytes of data and can also be used for edge computing.
* **AWS Snowmobile**: A massive truck-sized container used for exabyte-scale data transfers, typically used by big companies moving entire data centers.

These devices help you transfer data quickly, securely, and cost-effectively to AWS, especially when internet speed or reliability is an issue.

**Amazon S3 Storage Gateway**

It is a **hybrid cloud storage** service that connects on-premises environments to cloud storage in Amazon S3. It helps extend your local storage to the cloud by acting as a bridge.

**------------------------------------ AWS RDS(Relational Data Base) ----------------------------**

**AWS RDS (Amazon Relational Database Service)** is a **managed service** by AWS that makes it easy to set up, operate, and scale **relational databases in the cloud**.

**Purpose**: Handling administrative tasks like backups, patching, monitoring, and scaling. etc

**Database Creation Method:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Easy Create** | **Standard Create** |
| **Purpose** | Quick setup with **default settings** | Full control with **custom configurations** |
| **Use Case** | Best for beginners or quick testing | Best for production, critical workloads |
| **Configuration Options** | Minimal input (e.g., DB name, engine) | Full set of options (VPC, subnet group, backups, monitoring, encryption, etc.) |
| **Customization** | **Limited**, AWS chooses recommended defaults | **Fully customizable**: you can choose everything manually |
| **Launch Speed** | Faster, fewer steps | Slower, more detailed setup process |

**• Create a database. And execute commands on the EC2 instance :**

**• sudo yum install docker -y:** to install docker

**• sudo systemctl start docker:** to start the Docker service

**• sudo docker pull philippaul/node-mysql-app:02 :** Pulls (downloads) a Docker image named node-mysql-app with the tag 02 from Docker Hub (uploaded by user philippaul).

**• Sudo docker images**: to check the list of images

**•** sudo docker run --rm -p 80:3000 -e DB\_HOST="database-1.cdia6cyakrtn.ap-south-1.rds.amazonaws.com" -e DB\_USER="admin" -e DB\_PASSWORD="rootroot" -d philippaul/node-mysql-app:02

**•** You can pass **environment variables,** and **Spring Boot** can pick them up using ${} placeholders.

**Connect DB using EC2:**

sudo docker run -it --rm mysql:8.0 mysql -h dbhostname -u admin -p

**RDS Features:**

|  |  |
| --- | --- |
| **Feature** | **Description** |
| **Fully Managed Service** | Automates database setup, patching, backups, and failover. |
| **Multi-AZ Deployments** | High availability with automatic failover to a standby instance. |
| **Automated Backups** | Daily snapshots and point-in-time recovery (up to 35 days). |
| **Read Replicas** | Scales read workloads and improve performance for supported engines. |
| **Storage Auto-Scaling** | Automatically increases storage based on usage without downtime. |
| **High Performance Storage** | SSD-backed options (gp3/io1) with customizable IOPS. |
| **Security & Encryption** | Data encrypted at rest and in transit; VPC isolation; IAM authentication. |
| **Monitoring & Insights** | Integrated with CloudWatch and Performance Insights for deep diagnostics. |
| **Multi-Engine Support** | Choice of MySQL, PostgreSQL, MariaDB, Oracle, SQL Server, and Aurora. |

**Steps to Integrate Amazon S3 with Spring Boot**

1. **Add AWS SDK dependency** to your pom.xml or build.gradle.
2. **Set up AWS credentials**:
   * Use ~/.aws/credentials file **or**
   * Use environment variables.
3. **Create an S3 configuration class** to initialize S3Client.
4. **Create a service class** to handle file operations (upload, download, delete).
5. **Build a REST controller** to expose endpoints for file uploads.
6. **Add bucket name and region** to application.properties if needed.
7. **Run and test the API** using Postman or curl.

--------------------------- Amazon DynamoDB:------------------------------

**Amazon DynamoDB** is a **fully managed NoSQL database service** provided by AWS. It's designed for **high-performance**, **scalable**, and **serverless** workloads that require **low-latency** access to data at any scale.

**Key Features of DynamoDB:**

|  |  |
| --- | --- |
| **Feature** | **Description** |
| **NoSQL Database** | Key-value and document-based storage — schema-less and flexible. |
| **Fully Managed** | No need to manage servers, clusters, or patching. |
| **High Performance** | Millisecond response times, even at millions of requests per second. |
| **Scalable** | Automatic scaling for throughput and storage. |
| **Serverless** | Pay-per-request pricing and auto-scaling; no infrastructure to manage. |
| **Multi-Region & Global Tables** | Replicate tables across regions for high availability and low latency. |
| **Fine-Grained Access Control** | Integrates with IAM for secure, role-based access. |

**--------------------AWS Lambda Function -------------------------------**

**AWS Lambda** is a **serverless compute service** provided by AWS.  
It allows you to **run code without provisioning or managing servers**.  
You simply upload your code, and Lambda takes care of everything required to run and scale it.

**Event-Driven Execution:**

* Lambda is an event-driven service, meaning that it runs your code in response to certain triggers or events.
* These events can come from many different AWS services, like:
  + S3 (file uploads)
  + DynamoDB (database changes)
  + API Gateway (HTTP requests)
  + CloudWatch (scheduled events), etc.

**Steps**: Create a Function and write the function code in your in your programming language.

And an event on the above(S3, API Gateway). Ex. Send an email while the file uploads to the S3 bucket

**Limitations:**

|  |  |
| --- | --- |
| **Limitation** | **Description** |
| **Execution Time Limit** | Lambda functions can only run for a maximum of 15 minutes. If longer-running tasks are required, Lambda might not be the best choice. |
| **Stateless** | Lambda functions don’t keep state between invocations, making them best suited for tasks that don’t require long-term memory. |
| **Cold Start Delays** | If a Lambda function hasn’t run in a while, there may be a slight delay (called a "cold start") when it starts. This can add latency, but AWS offers ways to mitigate this for critical functions. |

**Advantage:**

|  |  |
| --- | --- |
| **Use Case** | **Description** |
| **Image Processing** | When users upload images, Lambda can automatically resize, compress, or apply filters to each image as it's uploaded. |
| **Data Transformation** | Lambda can clean or process data before storing it in a database, handling the transformation automatically. |
| **Real-Time Notifications** | Lambda can trigger notifications (email, SMS, etc.) instantly when an event occurs, such as a new user signing up. |

**Features:**

|  |  |
| --- | --- |
| **Feature** | **Description** |
| **Serverless** | No server management required; AWS automatically handles infrastructure. |
| **Event-Driven** | Lambda functions are triggered by events from other AWS services or HTTP requests. |
| **Auto-Scaling** | Automatically scales to handle varying loads and requests. |
| **Short Execution Time** | Functions can run up to 15 minutes per execution. |
| **Multiple Language Support** | Supports Node.js, Python, Java, C#, Go, Ruby, PowerShell, and custom runtimes. |
| **Integrated with AWS** | Easily integrates with S3, DynamoDB, API Gateway, CloudWatch, and more. |
| **Pay-per-Use** | Pay only for the compute time you use, billed in 100ms increments. |
| **Stateless** | Each function execution is isolated and does not retain state between invocations. |
| **Event Sources** | Can be triggered by AWS services like S3, DynamoDB, CloudWatch, API Gateway, etc. |
| **Cold Start** | When a function hasn't run recently, it may experience a slight delay (cold start). |

**------------------AWS Cloud Formation------------------------**

**AWS CloudFormation** is a service that helps you model and set up **AWS resources** so that you can spend less time managing infrastructure and more time focusing on your applications. It allows you to define your cloud infrastructure as **code** and deploy it consistently and automatically.

|  |  |
| --- | --- |
| **Feature** | **Description** |
| **Infrastructure as Code (IaC)** | Define your entire infrastructure using JSON or YAML templates, making it reproducible and version-controlled. |
| **Automated Provisioning** | Automatically create, modify, and delete AWS resources based on the template. |
| **Resource Dependencies** | Automatically handles dependencies between resources, ensuring that they are created in the correct order. |
| **Supports Many AWS Services** | Supports a wide range of AWS services, including EC2, S3, RDS, Lambda, and more. |
| **Cross-Region and Cross-Account Support** | Allows you to deploy & manage stacks in different regions or across multiple AWS accounts. |
| **Rollback on Failure** | Automatically rolls back changes if there’s an error during the creation or update process. |
| **Stack Outputs** | Outputs enable you to export values from one stack & use them in another stack/application. |
| **Custom Resources** | Extend CloudFormation capabilities by adding custom logic, such as running Lambda functions or invoking external APIs. |

**Steps**:

* create a stack
* Specify template ex.

****

* By using this YML/JSON template code. It will create an EC2 instance.
* **Note**: all mentions of info should be correct. Otherwise, it will give an error. Ex. **imageId**
* It is region-specific, but using a template can create resources globally.
* We can **update/delete/create** using a template.
* Using CLI with a template, we can schedule automatically very easily.
* We can create multiple resources, a security group create & assign, firewall. Means we can do everything using a template, which we perform manually.
* If we delete the stack, every resource will be deleted. So, cleanup perform very easily.
* Using CLI: **aws cloudformation** create-stack **--stack-name** MyStackName **--template-body file:**//path/to/template.yaml

**Why should we use:**

|  |  |
| --- | --- |
| **Reason** | **Explanation** |
| **Automation** | Creates AWS resources automatically using templates (YAML/JSON). |
| **Consistency** | Same infrastructure setup across **dev, test, and production environments.** |
| **Easier Updates** | Just modify the template and update the stack; AWS handles changes. |
| **Rollback Protection** | Automatically rolls back resources if deployment fails. |
| **Dependency Handling** | Automatically manages resource order (e.g., security groups before EC2). |

**----------------------- Route 53-------------------------------**

**The Domain Registrar** should have **Route53 nameservers** updated if the domain is not purchased from AWS.

**Means Route53 provides some nameservers. We need to add them to GoDaddy.**

**Steps:**

|  |  |  |
| --- | --- | --- |
| **Step** | **Action** | **Details** |
| 1 | **Get EC2 Public IP** | Go to **EC2 Dashboard → Instances → Your Instance → Public IPv4 address** and copy it. |
| 2 | **Open Route 53** | Go to the **AWS Route 53 Console**. |
| 3 | **Hosted Zones** | Select your **domain name** under **Hosted Zones** (you must have an existing domain). |
| 4 | **Create Record** | Click on **Create Record** to map your domain to the EC2 instance. |
| 5 | **Record Settings** | **Record Name**: Leave blank (for root domain) or use subdomain (e.g., www).**Record Type**: A (IPv4 address).**Value**: Paste your EC2 Public IP.**TTL**: Leave default (300 sec). |
| 6 | **Save Record** | Click **Create records** to save it. |
| 7 | **Test** | Open your domain in a browser (it should point to your EC2 instance). |

**Global Website with Low Latency:**

• **Latency Routing** automatically routes users to the region that gives **lowest network latency**.

• Commonly used when your **application is deployed across multiple regions** for **better speed and experience**.

Steps:

1. Open **AWS Route 53 → Hosted Zones → Select your domain**.

2. Create a record with **Public IP or Load Balancer**. Select **Latency Routing** instead of Simple Routing.

3. Choose the **region** where your EC2 instance is hosted (e.g., ap-south-1 for Mumbai).

4. Default TTL (300 seconds) is fine for latency routing.

5. If you have EC2 in multiple regions, create **multiple latency records per region.**

**Health Checks:** We can create a healthy check for every region and configure the healthy check ID in the same region records.

**Health Checks** are used to route traffic only to healthy resources.

It automatically stops sending traffic to unhealthy instances and routes to healthy ones.

**Types of DNS records** in **AWS Route 53:**

|  |  |  |
| --- | --- | --- |
| **Record Type** | **Purpose** | **Example** |
| **A Record** | Maps domain name to **IPv4 address** | example.com → 192.0.2.1 |
| **AAAA Record** | Maps domain name to **IPv6 addressss** | example.com → 2001:db8::1 |
| **CNAME Record** | Alias one domain to another domain | www.example.com → example.com |
| **MX Record** | Mail Exchange, routes emails to mail servers | example.com → mail.example.com |
| **NS Record** | Lists **Name Servers** for a domain | ns-123.awsdns-45.com |
| **PTR Record** | Reverse DNS lookup (IP to domain) | 1.2.0.192.in-addr.arpa → example.com |
| **Alias Record (Route 53 only)** | AWS feature to **map domain to AWS resources** like S3, ELB, CloudFront without extra cost | example.com → myloadbalancer-1234.elb.amazonaws.com |

**------------------------------ AWS CloudFront CDN-------------------------**

**AWS CloudFront** is a Content Delivery Network (**CDN**) that speeds up the delivery of web content to users by **caching it at servers (edge locations)** close to them, improving load times and performance globally.

AWS CloudFront primarily caches static content like **images, CSS, JavaScript, and videos**. It can also cache dynamic content **(e.g., HTML or API responses) if configured with caching policies and headers.**

By default, sensitive or user-specific data and backend logic are not cached. Cache behavior is **controlled via TTLs, cache behaviors, and origin headers.**

**Steps to Set Up AWS CloudFront (CDN)**

* **Prepare Your Origin** (e.g., S3 bucket, EC2 instance, or Load Balancer)
* **Go to AWS CloudFront** in the AWS Management Console
* **Create a Distribution**

- Choose **Web** for HTTP/HTTPS content delivery

- Select your **origin source** (e.g., S3 bucket, EC2, or custom origin)

* Configure Cache Settings

- Set TTLs (Time-to-Live) for caching

- Add cache behaviors for different paths or content types

* **Enable SSL/HTTPS** (optional but recommended)

- Use AWS Certificate Manager (ACM) for custom domain SSL

* **Review and Create** the distribution
* **Wait for Deployment** (takes a few minutes to propagate)
* **Update DNS** to point to the CloudFront distribution domain name
* **Test the CDN** by accessing the CloudFront URL or your custom domain

**--------------------------- Virtual Private Cloud (VPC) -------------------**

**Amazon VPC** is like your **own private network inside AWS**.

It’s where you can place your resources (like **EC2 servers**, **databases**, etc.) and **control who can connect**, **how they connect**, and **what they can talk to**.

**Steps to create a VPC:**

1. Go to the **VPC Dashboard** in the AWS Console and click on Create VPC  
Choose **"VPC only"** or **"VPC + Subnets"** (wizard-based)

2. Set the **IPv4 CIDR block** (e.g., 10.0.0.0/16) — defines the IP range.

3. Create VPC

**What is a VPC CIDR Block (in simple terms)?**

A **CIDR block** (Classless Inter-Domain Routing) is the **range of IP addresses** you assign to your **VPC** — it defines the **"size" of your VPC network**.

**What is a Subnet (in simple terms)?**

A **subnet** is a **smaller part of your VPC** that lets you **organize and isolate resources** (like EC2, RDS, etc.) within your virtual network.

Think of a **VPC as a big house**, and **subnets are the rooms** inside that house.

• A **VPC** has a large range of IP addresses (CIDR block).

• You **divide that range into subnets** — smaller sections of the network.

**Note**: Resource will be created within subnets (Ex, EC2 instance).

We create a Frontend app in the public subnet(Internet access) and Database in the private subnet.

**What is a Route Table?**

A **Route Table** is a **set of rules** (called **routes**) that decide **how traffic is directed** within your AWS network. Each subnet in your VPC must be **associated** with a route table, which controls the routing for that subnet.

• Every subnet is **associated** with a route table.

• **Public subnets** need a route to an **Internet Gateway**.

• **Private subnets** may use a **NAT Gateway** to access the internet **outbound only**.

**What is NAT Gateway?**

A **NAT Gateway** (Network Address Translation Gateway) allows resources in a **private subnet** to **access the internet** (e.g., for software updates), **without allowing inbound traffic** from the internet.

**What is VPC Peering in AWS?**

**VPC Peering** allows you to **connect two VPCs** so that they can **communicate with each other** using **private IP addresses** — just like they're on the same network.

Ex. Communication between two different regions server.

**What is a VPC Endpoint in AWS?**

Using a VPC Endpoint, we can connect to AWS services from within a VPC without using the public internet.

**What is an Elastic IP Address in AWS?**

An **Elastic IP address (EIP)** is a **static, public IPv4 address** that you can **allocate to your AWS account** and **associate with EC2 instances** or other resources.

• Normally, when an EC2 instance is stopped and restarted, its **public IP changes**.

• An **Elastic IP** stays **the same**, even if you stop, start, or replace the instance.

**What are VPC Flow Logs in AWS?**

**VPC Flow Logs** capture information about **IP traffic** going to and from network interfaces in your **VPC**. They help you **monitor**, **troubleshoot**, and **secure** your VPC network.

VPC Flow Logs are like **CCTV logs** for your AWS network — they show **who is talking to whom**, and whether the traffic was **allowed or denied**.

**What is AWS Direct Connect?**

**AWS Direct Connect** is a **dedicated, private network connection** between your on-premises data center (or office) and AWS — **bypassing the public internet**.

Direct Connect is like setting up a **private fiber cable** from your company network **directly into AWS**, giving you **faster, more secure, and reliable** network performance.

**What is AWS Client VPN?**

AWS Client VPN lets your **remote users** (like employees working from home) securely **connect to your AWS VPC** as if they were in your office.

--------------------------AWS Amplify---------------------------

**AWS Amplify** is a **cloud-based service** by AWS to **easily build, host, and deploy full-stack web and mobile applications** — without managing backend servers.

|  |  |
| --- | --- |
| **Feature** | **Description** |
| **Frontend Hosting** | Instantly deploy React, Angular, Vue, or static sites (HTML/CSS/JS) with Git integration (GitHub, GitLab, CodeCommit). |
| **Backend Integration** | Easily add backend features like authentication, APIs (GraphQL/REST), storage (S3), databases. |
| **CI/CD Pipeline** | Automatically build, test, and deploy code on every Git push. |
| **Authentication** | Integrates with Amazon Cognito for user login/signup (social logins also supported). |
| **Data & API** | Create backend APIs using AWS AppSync (GraphQL) or API Gateway (REST). |
| **Hosting** | Global, secure, fast hosting via AWS CloudFront (CDN). |