AWS

Day -1

1. What is a cloud?

Cloud computing is the on-demand delivery of IT resources—such as compute power, storage, databases, networking, and more—over the Internet, billed on a pay-as-you-go basis. Rather than owning and maintaining physical data centers, organizations tap shared infrastructure managed by a provider, enabling rapid provisioning and cost efficiency.

2. What is the difference between public vs. private cloud?

- **Public cloud:** Resources (servers, storage, networking) are owned and operated by a third-party provider and delivered over the Internet. It's a multi-tenant model—many customers share the same infrastructure—and you pay only for what you use.

- **Private cloud:** Infrastructure is dedicated to a single organization, either hosted on-premises or in a private data center. You maintain full control over hardware, security, and compliance, but bear the costs of purchase, operation, and upkeep.

3. Why is public cloud so popular?

- **Cost efficiency:** Converts large up-front capital expenditures into manageable operational expenses.

- **Scalability & flexibility:** Instantly scale resources up or down to match demand.

- **Global reach:** Deploy applications in multiple regions worldwide with minimal setup.

- **Managed services:** Outsources infrastructure maintenance—patching, backups, hardware refreshes—freeing teams to focus on innovation.

4. Why AWS?

Amazon Web Services (AWS) is the world's most comprehensive and widely adopted cloud platform, offering over 200 fully featured services from a global network of data centers. Millions of customers—startups, enterprises, and government agencies—choose AWS for its:

- **Breadth of services:** Compute, storage, databases, analytics, AI/ML, IoT, and more.
- **Global infrastructure:** 30+ geographic regions and 90+ Availability Zones for low-latency, resilient deployments.
- **Innovation pace:** Frequent service launches and feature updates.
- **Security & compliance:** Robust controls, certifications, and shared-responsibility model.

---

5. What are the trends of people moving back to private cloud?

Many enterprises are now "repatriating" workloads—shifting some applications from public to private cloud or on-premises—to optimize costs, improve data sovereignty, meet compliance needs, and boost performance. According to a 2024 Barclays CIO survey, **83% of CIOs** plan to repatriate at least some workloads this year, up from 43% in 2020. Organizations often adopt a balanced hybrid-cloud model rather than a full reversal

Day -2 AWS IAM

Q1: What is AWS IAM?

A: AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources. You use IAM to create and manage AWS users, groups, roles, and to define policies that grant permissions to those identities.

Q2: What is an IAM user?

A: An IAM user is an entity that represents a person or application. Each user has long-term credentials (a username/password or access keys) and inherits any policies

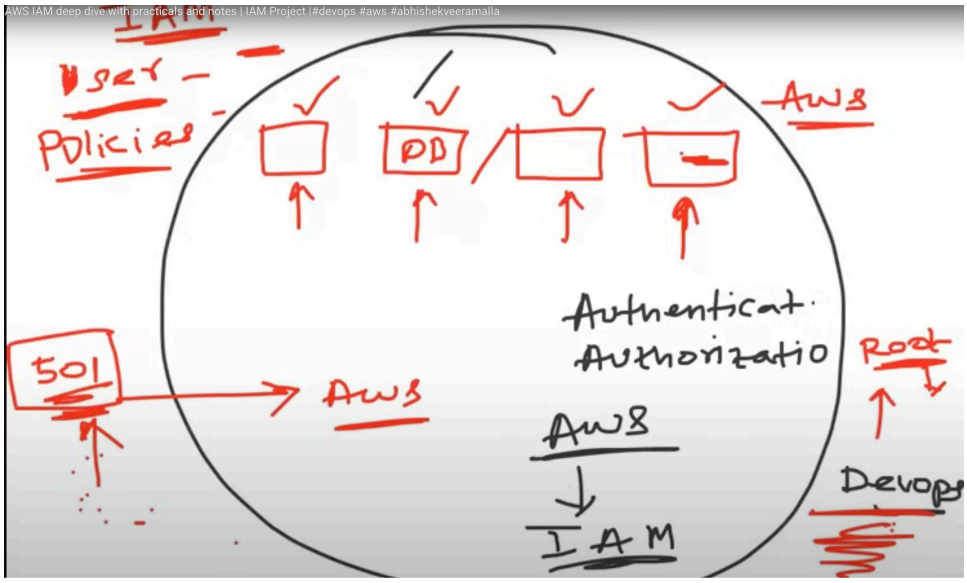
attached directly or via groups, allowing them to authenticate and perform actions in your AWS account.

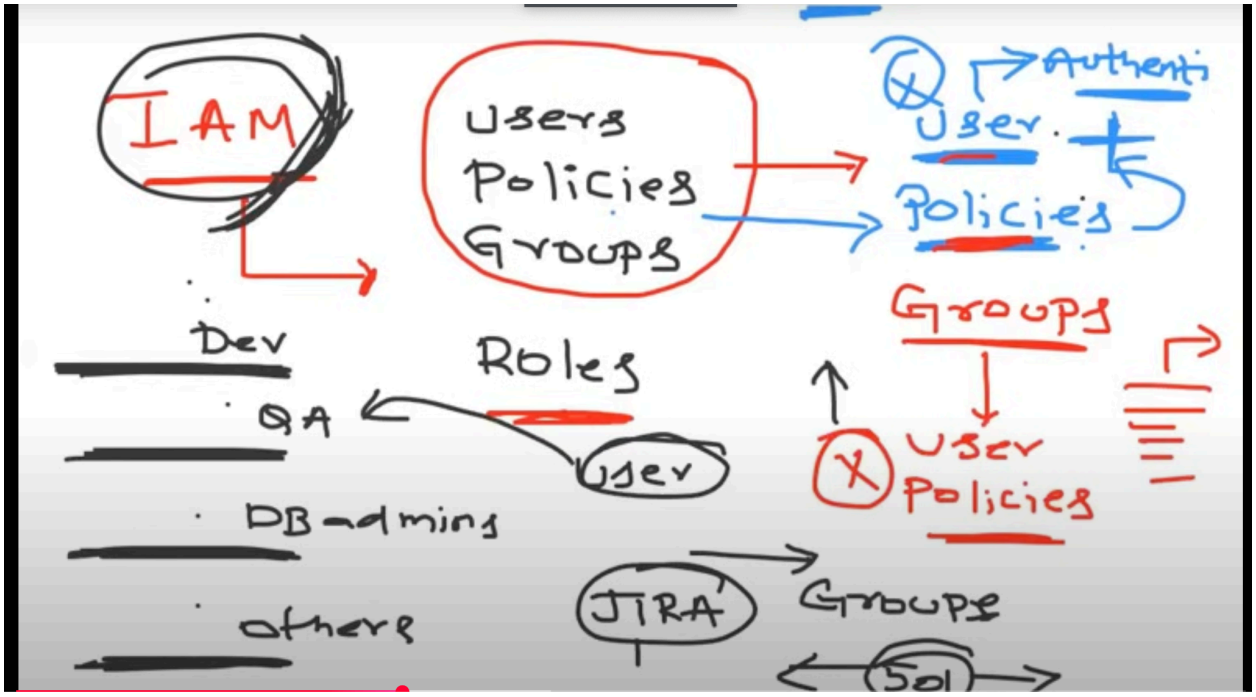
Q3: What is an IAM group?
A: An IAM group is a collection of IAM users. You attach policies to the group, and every user in that group automatically inherits those permissions—simplifying permission management for teams with similar roles.

Q4: What is an IAM role?
A: An IAM role is a set of permissions that can be assumed by trusted entities—such as IAM users, applications, or AWS services—using temporary security credentials. Roles let you delegate access without sharing long-term credentials.

Q5: What is an IAM policy?
A: An IAM policy is a JSON document that defines permissions. When you attach a policy to an IAM identity (user, group, or role) or an AWS resource, it specifies which actions are allowed or denied on which resources, and under what conditions.

Q6: What is the principle of least privilege?
A: The principle of least privilege means granting identities (users, roles) only the permissions they need to perform their tasks—and no more. This minimizes the risk of accidental or malicious actions.





Day -3 EC2

Q1: What is an EC2 instance?
A: An Amazon EC2 (Elastic Compute Cloud) instance is a virtual server in AWS's cloud, providing resizable compute capacity on which you can run applications.

Q2: What is an AMI (Amazon Machine Image)?
A: An AMI is a template that contains the OS, application server, and applications to launch new EC2 instances with a preconfigured environment.

Q3: How do instance types differ?
A: Instance types bundle CPU, memory, storage, and networking capacity for different workloads (e.g., compute-optimized, memory-optimized, general purpose).

Q4: What is an EBS volume?
A: An EBS (Elastic Block Store) volume is network-attached block storage you can mount to one EC2 instance, persisting independently from the instance's lifecycle.

Q5: What is a key pair?
A: A key pair consists of a public key (stored in AWS) and a private key (you download). You use the private key to securely SSH into Linux instances.

Q6: What is a security group?
A: A security group is a virtual firewall attached to instances. It's **stateful**, letting you define inbound and outbound rules by protocol, port, and source/destination.

Q7: What is a network ACL?
A: A network ACL is a **stateless** firewall applied at the subnet level; rules are evaluated in order and must explicitly allow return traffic.

Q8: How do you SSH into an EC2 instance?
A:

1. Ensure the instance has a public IP and its security group allows port 22 inbound.

2.Run:

bash
CopyEdit
ssh -i /path/my-key.pem ec2-user@ec2-x-x-x-x.compute-1.amazonaws.com

Q9: What's the difference between "stop" and "terminate"?

- **Stop:** Shuts down the instance; root EBS volume persists (you can restart). Billing for the instance stops, but you pay for the attached EBS.

- **Terminate:** Shuts down and deletes the instance; by default, root EBS is deleted.

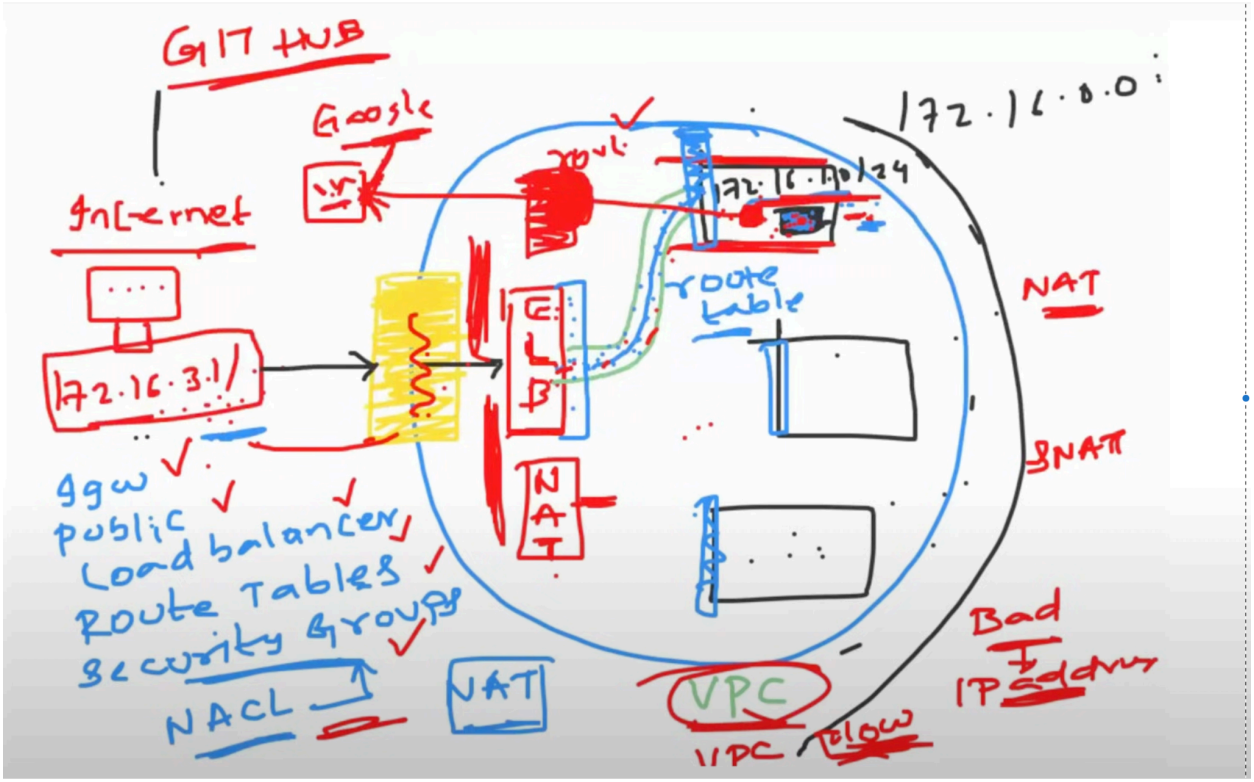
Q10: Basic web-app deployment on EC2

1. Launch an EC2 instance using a Linux AMI.

2. Open ports 22 (SSH) and 80 (HTTP) in its security group.

3. SSH in, install your web server (e.g., Apache or Nginx).
4. Deploy your app files to /var/www/html.
5. Visit the instance's public DNS in a browser.

Day -4 VPC (Virtual Private Control)



Q1: What is a VPC?
A VPC (Virtual Private Cloud) is a logically isolated virtual network you define within AWS. It closely resembles a traditional on-premises network, but leverages AWS's scalable infrastructure. You choose its IP address range, create subnets, configure route tables, and attach gateways to suit your application needs.

Q2: What is a subnet, and how do public and private subnets differ?
A *subnet* is a range of IP addresses within your VPC (each resides in one Availability Zone).

- Public subnet: has a route in its route table to an Internet Gateway, so resources launched there can send and receive traffic from the internet.

- Private subnet: has no direct route to the Internet Gateway (or routes only to a NAT device), so instances can reach out (e.g., for updates) but inbound connections from the internet are blocked.

Q3: What is an Internet Gateway (IGW)?
An Internet Gateway is a horizontally scaled, redundant AWS component attached to your VPC that allows resources in public subnets to communicate with the internet. Without an IGW and a matching route in the subnet's route table, instances cannot access or be accessed from the internet.

Q4: What is a NAT Gateway (and how does it differ from a NAT instance)?

- A NAT Gateway is an AWS-managed, highly available service placed in a public subnet that enables instances in private subnets to initiate outbound IPv4 (or IPv6 with DNS64/NAT64) traffic to the internet, but prevents unsolicited inbound connections.

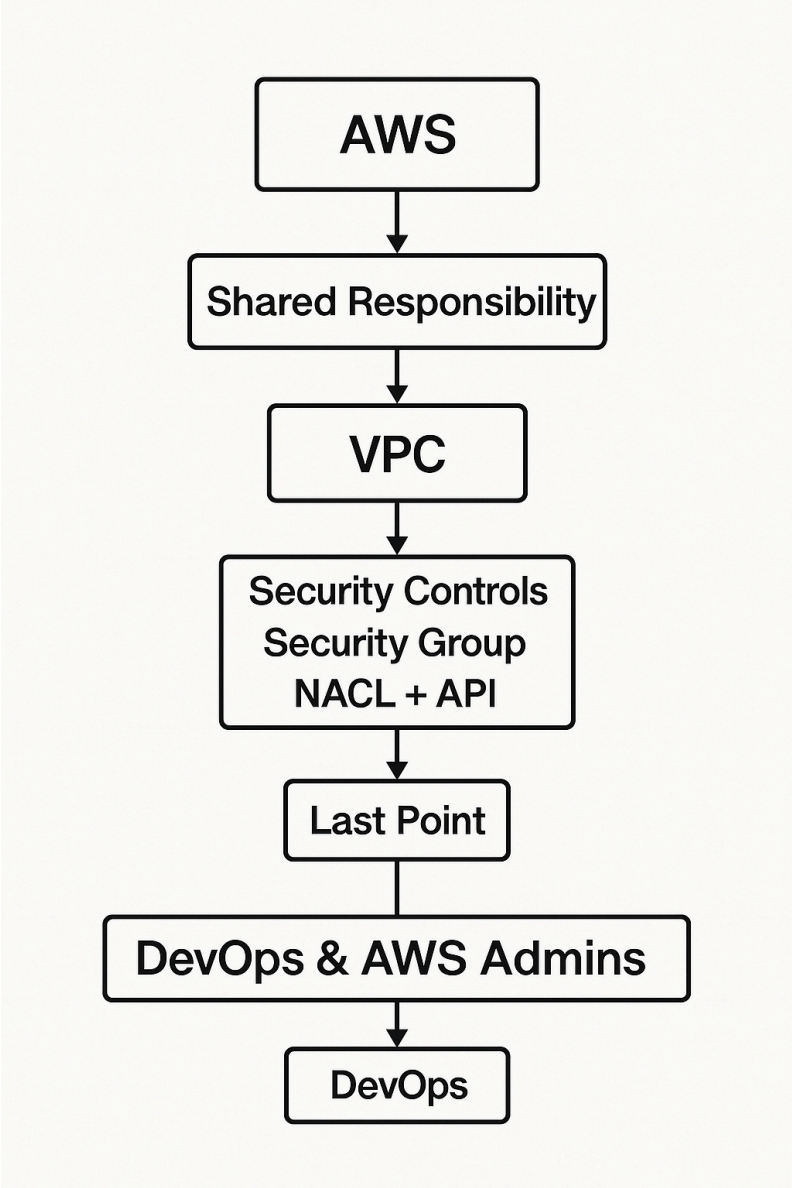
- A NAT instance is an older, self-managed EC2 instance configured to perform NAT. Unlike NAT Gateways, NAT instances require you to manage scaling, availability, and software updates.

Q5: What are route tables?
A route table contains rules (routes) that determine where network traffic from subnets is directed. Each subnet is explicitly or implicitly associated with one route table. Routes specify a destination CIDR and a target (e.g., igw-, nat-, vgw-, or peering connection). By customizing route tables, you control traffic flow between subnets, to the internet, or to on-premises networks

Day -5 AWS Security Group and NACL

NACL - Automation for security group, subnet level

Security at EC2 instance level - security group



Security Groups

Workflow

1. **Inbound Web Request**
    - A user browses to your public IP or domain.
    - Request hits the Internet Gateway and is routed to the EC2 instance's subnet.
    - **Security Group** checks inbound rules (ports 80/443) and allows the request to reach the EC2 instance.
2. **Application Processing**
    - The EC2 instance's application processes the request (e.g., a web page or API call).
3. **Outbound Calls**
    - If your application needs to fetch external data (for example, calling an external API or downloading updates), it sends traffic out.
    - **Security Group** outbound rule (allow all) permits this.
    - The **Route Table** sends traffic either to the Internet Gateway (if the instance has a public IP) or to a NAT Gateway (if it's in a private subnet).
4. **SMTP/E-mail Flow**
    - If you've added port 25 to your SG, your instance can send or receive mail via SMTP.
    - Without that rule, SMTP would be blocked inbound by default SG behavior.
5. **Return Traffic**

- Responses from the internet return via the IGW/NAT Gateway.

- Because SGs are *stateful*, return traffic to an allowed outbound or inbound request is automatically permitted—even if no explicit rule for that return port exists.

Route tables operate at the subnet level, while Security Groups are applied at the instance level.

You can layer in Network ACLs on your subnets for stateless, allow-and-deny rules, but they weren't shown in our simplified diagram.

NACL

Workflow of a Network ACL (NACL)

1. Packet Arrives at the Subnet Boundary
   – Traffic entering or leaving a subnet first hits the NACL associated with that subnet.

2. Inbound Evaluation

   - Rules are evaluated in ascending order by rule number.

   - The first matching rule (allow or deny) applies; if no rule matches, the default "deny all" catches it.

   - If allowed, the packet moves on to the subnet's route table (then your Security Group and instance).

3. Outbound Evaluation

- Responses or outgoing requests from instances go back through the same NACL, using its outbound rule set.

- Again, rules are checked by number, first match applies, then default deny.

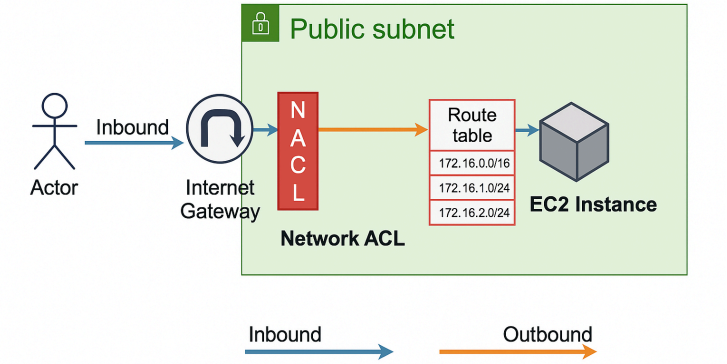
4. Stateless Behavior

    - NACLs are stateless: return traffic must be explicitly allowed by the NACL's opposite direction rules (there's no automatic return-traffic allowance).

Feature	**Network ACL (NACL)**	**Security Group (SG)**
Level of Application	Subnet-level	Instance-level
Statefulness	Stateless (must explicitly allow both directions)	Stateful (return traffic auto-allowed)
Rule Types	Allow **and** Deny rules	Allow-only rules (implicit deny otherwise)
Evaluation Order	By rule number (lowest → highest), first match wins	All rules evaluated; if any allow, traffic passes

Default Rules	Implicit “Deny all inbound & outbound”	Implicit “Allow all outbound”; “Deny all inbound except itself”
Use Cases	Broad subnet-wide controls, deny specific IPs or ports	Fine-grained per-instance access control
Rule Limit	Up to 20 rules per direction (in & out)	Up to 60 rules per group

NACL act as the first layer of defence



Practical I did

1. Create Your Own VPC

- Pulled up the VPC wizard and defined your own CIDR block (e.g. 10.0.0.0/16).
- **Why it matters:** A VPC is your private slice of AWS’s network. Nothing can talk to your instances until you explicitly open up paths.

---

2. Make a Subnet & Attach an Internet Gateway

- Created a subnet in that VPC and attached an Internet Gateway (IGW), then updated the subnet's route table so that `0.0.0.0/0 → IGW`.

- **What's happening:**

    - **Subnet + Route Table** together decide "which way does traffic go?"

    - The IGW makes sure traffic to/from the public Internet can cross the VPC boundary.

3. Launch Your EC2 Instance in That Subnet

- Launched an instance, made sure it got "Auto-assign Public IPv4," and put it in your new subnet.

- **Why:** With a public IP and the right route, your instance can now (potentially) send and receive Internet traffic.

4. Default Network ACL (NACL) Checks Traffic First

- **Out of the box:** Every subnet gets a NACL that **allows all** inbound & outbound (rule 100: ALLOW all).

- Your traffic wasn't blocked here—the NACL was wide open by default.

- **Tip:** NACLs are **stateless**—if you did lock something down here, you'd need matching allow-rules in both directions.

5. Default Security Group (SG) Next Filters Traffic

- **Out of the box:** The “default” SG allows:
    - **Inbound:** only from itself (i.e. other members of that SG)
    - **Outbound:** anywhere
- **What you experienced:**
    - You could SSH in on port 22 (you probably added that rule earlier), but **port 8000 was blocked**—no inbound rule yet.

6. Install & Run Your Python App on Port 8000

- **You did:** Connected over SSH, installed Python, launched your web app listening on port 8000.
- **Result:** The app was running, but **nobody on the Internet could see it**—the SG was still blocking 8000.

7. Open Port 8000 in the Security Group

- **You did:** Edited the SG’s **Inbound rules**, added “Custom TCP | Port 8000 | Source 0.0.0.0/0.”
- **What changed:**
    - Now the SG says “**ALLOW** any Internet IP to connect on TCP 8000.”
    - Since the NACL was already allowing it, your traffic flows right through.

8. What’s Actually Happening When Someone Hits `http://your-public-ip:8000`

1. **User → Internet Gateway**

- Packet enters your VPC at the IGW.

2. **Internet Gateway → NACL**
    - NACL checks its inbound rules (default "allow all") → **passes**.
3. **NACL → Route Table**
    - Route table sees "10.0.1.0/24 (your subnet) → local" → forwards into the subnet.
4. **Route Table → Security Group**
    - SG checks its inbound rules—now sees "ALLOW TCP 8000 from 0.0.0.0/0" → **passes**.
5. **Security Group → EC2**
    - Your Python app on port 8000 finally gets the packet and responds.
6. **Return Packets**
    - SG is **stateful**, so it automatically allows the response back out—even though you didn't create an explicit outbound rule for that connection.
    - NACL (stateless) still allows the return because its outbound rules are wide open.

Day -6 Route 53

When you Route 53, it provides DNS as Service. DNS is the one that maps your domain name with your IP address.

1. What Is DNS?

- **Domain Name System (DNS)** translates human-friendly names (e.g. `www.example.com`) into IP addresses that computers use to route traffic.

- Think of it as the Internet's phonebook: you look up a name, get a number, and then call.

2. Route 53 Overview

- **Amazon Route 53** is AWS's highly available, scalable DNS web service.

- It provides:

    - **DNS resolution** (public & private)

    - **Domain name registration**

    - **Health checks & failover routing**

3. Domain Registration

- **Registering a Domain**: You can buy and manage top-level domains (TLDs) directly through Route 53 (e.g., `.com`, `.org`).

- **WHOIS & Renewal**: Route 53 keeps your contact info in WHOIS, handles renewals, and can auto-renew domains to prevent expiration.

- **Transfer**: Move existing domains from other registrars into Route 53 for unified billing and management.

4. Hosted Zones

- A **Hosted Zone** is your container for DNS records for a domain. There are two types:

    1. **Public Hosted Zone**: DNS records accessible over the internet.

2. **Private Hosted Zone**: DNS records only resolvable within one or more specified VPCs.

- When you create a Hosted Zone, Route 53 automatically assigns a set of authoritative name servers.

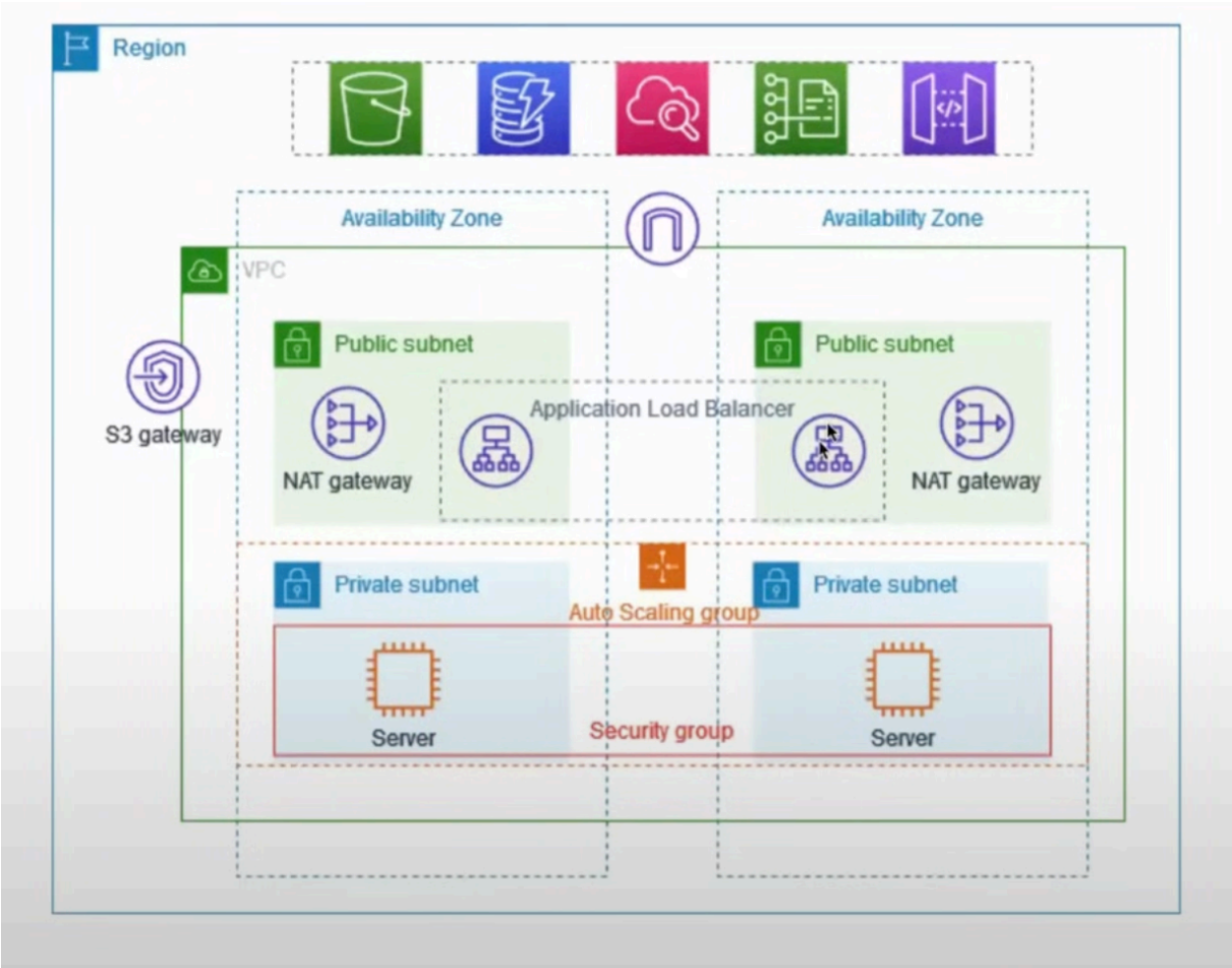
5. Record Sets (DNS Records)

- Inside each Hosted Zone you create **record sets** that map names to targets:
    - **A / AAAA**: domain → IPv4 / IPv6 address
    - **CNAME**: alias → another domain name
    - **MX**: mail-exchange servers
    - **TXT**: arbitrary text (e.g. SPF, verification)
    - **ALIAS**: AWS-specific pointer to ELB, CloudFront, S3 website endpoints, etc.

6. Health Checks & Failover

- **Health Checks** continuously poll your endpoints (HTTP, HTTPS, TCP) to verify service health.
- **Routing Policies** allow you to react to health check status:
    - **Simple**: single record, no health check
    - **Failover**: primary & secondary endpoints—Route 53 automatically redirects if the primary fails
    - **Weighted / Latency / Geolocation** (advanced routing) can also be tied to health checks for sophisticated traffic management.

Day -7 VPC with public-private subnet in production.



In AWS, a bastion host is an EC2 instance that serves as a secure gateway, allowing users to access private instances within an AWS Virtual Private Cloud (VPC) from outside the network. It acts as a central point of access, minimizing the risk of exposing private resources directly to the internet.

Day -8 Interview Questions

1) You have been assigned to design a VPC architecture for a two-tier application. The application needs to be highly available and scalable. How would you design the VPC architecture?

   Ans) Availability issue is solved by using more availability zones, and the scalable issue is solved by using security groups. I'd set up a VPC that spans at least two Availability Zones with public subnets hosting an Internet-facing load balancer and private subnets running an Auto Scaling group of application servers, use an Internet Gateway (plus NAT Gateways in each AZ) for the right traffic flow, and rely on Security Groups to lock down exactly which ports and sources can reach your load balancer and app servers—giving you both high availability and automatic scalability in one simple design.

2) Your organization has a VPC with multiple subnets. You want to restrict outbound internet access for resources in one subnet but allow outbound internet access for resources in another subnet. How would you achieve this?

   Ans) **To control outbound internet access per subnet, use two route tables:**

- **Case 1 – No outbound internet (isolated subnet):**
  Attach a route table that **omits** the `0.0.0.0/0 → Internet Gateway` route. Resources in this subnet cannot reach the Internet.

- **Case 2 – Outbound internet allowed (public subnet):**
  Attach a route table that **includes** the `0.0.0.0/0 → Internet Gateway` route. Resources in this subnet can access the Internet.

Feature	NAT Gateway	Bastion Host
Primary Purpose	Enables **outbound** Internet access from private-subnet instances	Provides secure **inbound** SSH/RDP access into private-subnet instances

Traffic Direction	Private → Internet (and back)	Internet → Bastion → Private
Managed Service	✅ Fully managed by AWS	❌ You must launch and maintain an EC2 instance
High Availability	✅ Built-in HA within an AZ (can deploy one per AZ for zone-redundancy)	❌ Only as HA as you configure (e.g., multiple bastions, ELB)
Scalability	✅ Automatically scales to handle traffic	❌ Must size and scale EC2 instance(s) yourself
Security	– Controlled via route tables and Security Groups	– Controlled via tight SG (e.g. limit SSH to your IP), plus OS patching
Cost Model	Charged per hour + data processed	Charged as EC2 instance (per hour + data)
Use Case	Private servers need internet (OS updates, API calls) without public IPs	Administrators need shell/desktop access to private servers

3) you have a VPC with a public subnet and a private subnet. Instance of the private subnet need to access the internet after the software update. How would you allow internet access for instances in a private subnet?

Ans) Place a NAT Gateway in your public subnet (with an Elastic IP), then point the private subnet's route table `0.0.0.0/0` route to that NAT Gateway—this lets your

private instances pull updates or call external APIs without ever exposing them to inbound Internet traffic.

4) You have launched EC2 instances in your VPC, and you want them to communicate with each other using private IP addresses. What steps would you take to enable this communication?

   Ans) Ensure both instances are in the **same VPC** (or peered VPCs) so the default route table's local route handles private-IP traffic, then open up their **Security Groups** to allow the needed ports from each other's SG (e.g. SG-A inbound from SG-B and vice versa). With Network ACLs left at their default "allow," the two instances can now talk directly over their private IPs.

5) You want to implement strict network access control for your VPC resources. How would you achieve this?

   Ans) With the help of NSCL and security groups.

6) Your organization requires an isolated environment within the VPC for running sensitive workloads. How would you set up this isolated environment?

   Ans) Use a private subnet. I'd put those workloads into a dedicated private subnet that has **no** `0.0.0.0/0 → Internet Gateway` or NAT-Gateway route, lock it down with a tight Network ACL and Security Group that only allows the bare-minimum VPC traffic, and—if the servers still need to call AWS services (S3, KMS, Secrets Manager, etc.)—attach the appropriate VPC Endpoints. This gives you a fully isolated network enclave: no inbound or outbound Internet, only controlled internal and AWS-service access.

7) Your application needs to access AWS services such as S3 securely within your VPC. How would you achieve this?

   Ans) Use VPC endpoints. I'd create a **VPC Endpoint** for S3 (a Gateway Endpoint) inside my VPC and associate it with the private subnet's route table—this lets my instances call S3 over AWS's internal network (no Internet involved). To lock it down further, I'd update the S3 bucket policy to only allow access via that VPC Endpoint.

8) What is the difference between an ACL and a subnet? Explain with a use case?

Ans) A **subnet** is simply a range of IP addresses within your VPC (tied to one Availability Zone) used to place and isolate your resources, while a **Network ACL (Access Control List)** is a stateless, subnet-level firewall that lets you explicitly allow or deny traffic in and out of those subnets.

Use case: You might put your databases in a **private subnet** so they're never directly reachable from the Internet, then attach a **NACL** to that subnet which **denies all inbound** traffic except on port 3306 from your application-server subnet and **allows only** outbound traffic to your NAT Gateway for patching—giving you a second layer of security at the subnet boundary.

9) What is the difference between IAM users, group roles, and policies?

   Ans) IAM users are permanent identities with long‑term credentials (passwords or access keys) that represent individual people or applications needing direct access to AWS. To simplify permissions, you can group users into IAM groups—any policy attached to a group automatically applies to all its members—so you don't have to manage each user's rights one by one. IAM roles, by contrast, are temporary identities that you "assume" to gain short‑lived credentials; they're ideal for granting access to AWS services (like when an EC2 instance needs to read from S3) or for cross‑account access without sharing long‑term keys. Finally, IAM policies are the JSON documents that actually define which actions are allowed or denied; you attach policies to users, groups, or roles to control precisely what each identity can do in your AWS account.

10) You have a private subnet in your VPC that contains a number of instances that should not have a direct internet access. However, you still need to be able to securely access these instances for administrative purposes. How would you set up a bastion host to facilitate this access?

    Ans) I'd launch a small, hardened EC2 instance in a **public subnet** (your bastion host), give it an Elastic IP, and lock down its Security Group to allow SSH (TCP 22) **only** from your office (or VPN) IP range. Then, in your **private subnet**, update the application‑server Security Group to permit SSH **only** from the bastion's Security Group—no other inbound SSH. Now, administrators SSH into the bastion's public IP, and from there SSH over the private network to the private

instances. This keeps your private servers off the Internet while still letting you manage them securely through a single, well-controlled jump box.

Day -9 S3

1. What Is Amazon S3?

Amazon Simple Storage Service (S3) is a highly durable, infinitely scalable object store—think of it as a key-value data store where you upload "objects" (files + metadata) into globally unique "buckets."

2. Buckets & Objects

- Bucket: A top-level container with a unique name in the S3 namespace.
- Object: The actual file you store, identified by a key (its filename/path) inside a bucket.

3. Storage Classes

Choose based on access patterns and cost trade-offs:

- Standard (frequent access)
- Intelligent-Tiering (automated cost optimization)
- Standard-IA, One Zone-IA (infrequent access)
- Glacier, Deep Archive (archival)

4. Access Control

- Bucket Policies (JSON docs on the bucket itself) and IAM Policies (attached to users/roles) let you grant or deny actions at scale.
- ACLs are the original, legacy grants you can apply per bucket or object (less flexible than policies).
- Pre-signed URLs give time-limited, scoped access to private objects. 5. Static Website Hosting

1. Enable Website Hosting under the bucket's Properties, set your Index (e.g., index.html) and Error (e.g., error.html) documents.

2. Disable Block Public Access at the bucket (and account) level so you can expose objects.

3. Add a Bucket Policy granting s3:GetObject on arn:aws:s3:::your-bucket/* to Principal: "*".

4. Use the Website Endpoint (http://your-bucket.s3-website.<region>.amazonaws.com) to serve your static site.

Key Takeaway:
S3 isn't just "storage" – it's a full-blown, durable object store with fine-grained permissions and built-in static website hosting, ideal for everything from backups to serving entire static web apps.

Day -10 AWS CLI

Command line Interface,Terraform, Cloud formation, CDK.

Command line Interface- python utility

What Is AWS CLI?

A unified, open-source tool that lets you manage AWS services from your terminal or automate them through scripts—equivalent to the AWS Console in command-line form

- AWS Documentation
  https://docs.aws.amazon.com/cli/latest/
-