CLOUD NETWORKING ON AWS

The AWS Global Network

- AWS operates a private global **backbone network** that connects its cloud regions and major service locations.
- A **Region** contains multiple isolated **Availability Zones**, giving AWS a built-in model for low-latency and fault-tolerant design.
- **Edge locations** and regional caches bring content and services closer to users for faster delivery.
- Network design in AWS starts with geography: where workloads run, where users connect, and how traffic moves between them.

Regions, AZs, and Edge

- An AWS **Region** is a separate geographic area where you deploy cloud resources and keep most application traffic local. Examples include **N. Virginia** (`us-east-1`), **Ohio** (`us-east-2`), **Oregon** (`us-west-2`), and **Frankfurt** (`eu-central-1`). d
- Each Region includes multiple **Availability Zones**, which are isolated data center groups connected by high-bandwidth, low-latency links. In `us-east-1`, example AZ names include `us-east-1a`, `us-east-1b`, and `us-east-1c`.
- **Edge locations** sit closer to end users and support services such as CloudFront, Route 53, and AWS Shield.
- A common pattern is to place applications across multiple AZs for resilience, such as running one tier in `us-east-1a` and another in `us-east-1b`, while using edge infrastructure to improve global reach and performance.

Amazon VPC Fundamentals

- An Amazon **VPC** is a logically isolated virtual network that you define inside an AWS Region.
- A VPC gives you control over core network settings such as **IP address ranges**, **subnets**, **route tables**, and **security policies**.
- Most application resources, such as EC2 instances and many managed services, are deployed into a VPC so they can communicate on private addresses.
- In practice, the VPC is the starting point for AWS network architecture: you create the address space first, then decide how traffic enters, leaves, and moves internally.

Hosts, Networks, and Subnets

- A **host** is any device or service with its own IP address, such as an EC2 instance, load balancer node, or on-premises server.
- A **network** is a collection of IP addresses that can be treated as one address block, while a **subnet** is a smaller subdivision of that block.
- A **router** or routing function moves packets between networks, while hosts communicate directly only within their local subnet unless traffic is forwarded.
- In AWS, a VPC is the larger network boundary, and each subnet is a smaller address range placed inside one Availability Zone.

CIDR and IP Planning

- **CIDR** notation defines an IP address block with a base address and prefix length, such as `10.0.0.0/16`.
- In AWS, the VPC CIDR block determines how much private address space you have available for all subnets and resources inside that VPC.
- Smaller prefix lengths give you more addresses: a `/16` is much larger than a `/24`, so address planning affects both current deployment and future growth.
- Good IP planning avoids overlapping ranges across VPCs, data centers, and VPN-connected networks, which becomes critical when you add peering, Transit Gateway, or hybrid links.

How CIDR Notation Works

- In `10.0.0.0/16`, the `/16` means the first 16 bits identify the network and the remaining bits are available for host addresses.
- A longer prefix means a smaller block: `10.0.1.0/24` is a subnet inside `10.0.0.0/16`, and it contains far fewer addresses.
- You can think of subnetting as borrowing bits from the host portion to create smaller networks, such as splitting one `/16` VPC into many `/24` subnets.
- In AWS, this matters because each subnet must fit entirely inside the VPC CIDR range and be large enough for the resources you plan to place there.

Subnets and Route Tables

- A **subnet** is a slice of the VPC CIDR block, and each subnet exists in exactly one Availability Zone.
- AWS commonly uses **public subnets** for internet-facing components and **private subnets** for internal application or database tiers.
- A **route table** is a set of rules that tells AWS where to send traffic for different destination ranges, such as local VPC traffic, the internet, or another network.
- In practice, a subnet becomes public or private based on its route table: if it has a route to an Internet Gateway, it can support internet-facing resources; if not, it stays internal.

Internet Access Patterns

- An **Internet Gateway** lets resources in a public subnet send and receive traffic to the public internet when they also have public IP addresses.
- A **NAT Gateway** lets instances in private subnets make outbound internet connections, such as for software updates, without accepting inbound connections from the internet.
- A common three-tier pattern puts load balancers in public subnets, application servers in private subnets, and databases in even more restricted private subnets.
- Not every workload needs internet access at all; many AWS architectures keep internal services private and reach AWS services through private networking features instead.

Security Groups and NACLs

- A **security group** is a virtual firewall attached to an instance or service network interface, controlling which inbound and outbound traffic is allowed.
- Security groups are **stateful**, which means return traffic is automatically allowed when you permit an incoming or outgoing connection.
- A **network ACL** acts at the subnet boundary and can allow or deny traffic based on rules for source, destination, and port ranges.
- In practice, AWS architects usually rely on security groups for most access control and use NACLs as a broader subnet-level filter when needed.

Reaching AWS Services Privately

- Many AWS applications need to access services such as S3, DynamoDB, or Systems Manager without sending traffic over the public internet.
- A **VPC endpoint** provides private connectivity from a VPC to supported AWS services while keeping traffic on the AWS network.
- **Gateway endpoints** are commonly used for services like S3 and DynamoDB, while **interface endpoints** create elastic network interfaces for private access to many other AWS services.
- Private service access improves security and simplifies architecture because resources in private subnets can reach AWS services without requiring public IP addresses or NAT for those connections.

Basic DNS Review

- **DNS** translates human-readable names such as `example.com` into IP addresses that clients can actually use.
- A DNS **record** maps a name to some information, such as an IPv4 address (`A` record), IPv6 address (`AAAA` record), or another hostname (`CNAME` record).
- DNS resolution is distributed: clients ask resolvers, resolvers query authoritative name servers, and the result is cached for some period of time.
- In cloud systems, DNS often determines which endpoint users reach, so it becomes part of both availability design and traffic management.

DNS Providers and Name Servers

- A **domain registrar** is the company where you buy a domain name, while a **DNS provider** is the service that hosts the DNS records for that domain.
- Sometimes the registrar and DNS provider are the same company, but they do not have to be; for example, you might buy a domain from one provider and manage DNS in AWS Route 53.
- The DNS provider gives you a set of authoritative **name servers**, and you configure your domain to use those name servers.
- Once that connection is in place, anyone looking up your domain will eventually reach the DNS provider you chose, and that provider will answer with your records.

DNS Records for a Personal Domain

- To publish a personal site, you first register a domain name, point it at your DNS provider's **name servers**, and then create records inside the hosted zone for that domain.
- An **A record** points a name such as `www.example.com` to an IPv4 address, while an **AAAA record** does the same for IPv6.
- A **CNAME record** points one hostname to another hostname, which is useful when `www.example.com` should follow the DNS name of an AWS load balancer or another managed service.
- In practice, you might point `example.com` or `www.example.com` at an EC2 instance's public IP, or more commonly at a load balancer in front of your AWS-hosted application; the **TTL** controls how long resolvers cache that answer before checking again.

DNS with Route 53

- Amazon **Route 53** is AWS's DNS service for registering domains, hosting DNS zones, and routing traffic to AWS or non-AWS endpoints.
- In Route 53, you usually create a **hosted zone** for your domain and then add records such as `A`, `AAAA`, `CNAME`, or AWS-specific **alias** records.
- An **alias record** lets you point a domain name at AWS resources such as an Application Load Balancer, CloudFront distribution, or S3 website endpoint without manually tracking IP addresses.
- Route 53 also supports routing policies such as weighted, latency-based, and failover routing, which makes DNS part of your application's availability and traffic-management strategy.

Private AWS Connectivity

- Some organizations need private network connections between AWS and on-premises data centers, offices, or colocation environments.
- An AWS **Site-to-Site VPN** creates an encrypted tunnel over the public internet, which is usually the faster and cheaper option to set up.
- **AWS Direct Connect** provides a dedicated private link into AWS, which can offer more consistent performance and is often used for larger or more predictable traffic volumes.
- These options are the foundation for **hybrid cloud** networking, where some systems stay on-premises while others run inside AWS VPCs.

Multi-VPC Design Patterns

- Many AWS environments use more than one VPC so teams can separate workloads, ownership, and security boundaries.
- An **application VPC** typically contains the resources for one system or product, such as web servers, application servers, and databases.
- A **shared services VPC** holds common infrastructure used by many workloads, such as directory services, logging systems, DNS, monitoring, or security tools.
- This design improves isolation and team autonomy, but it also creates a networking problem: the VPCs still need controlled ways to communicate with each other.

VPC Peering Tradeoffs

- **VPC peering** creates a private connection between two VPCs so resources can communicate using private IP addresses.
- Peering works well for simple one-to-one relationships, such as connecting an application VPC to a shared services VPC.
- The tradeoff is scale: if many VPCs need to talk to each other, the number of peering connections and route updates grows quickly.
- Peering is also **non-transitive**, which means if VPC A peers with VPC B and VPC B peers with VPC C, VPC A does not automatically reach VPC C through B.

Transit Gateway Design

- AWS **Transit Gateway** acts as a central network hub that connects multiple VPCs, VPNs, and Direct Connect attachments.
- Instead of building many point-to-point peering links, each VPC can attach once to the Transit Gateway and use it as a shared routing core.
- This hub-and-spoke model simplifies routing, scales better across many VPCs, and makes it easier to enforce central network policy.
- Transit Gateway is commonly used in larger AWS environments with multiple accounts, shared services, and hybrid connectivity back to on-premises networks.

Hybrid Network Architectures

- A **hybrid architecture** combines cloud resources in AWS with systems that remain on-premises, in a campus network, or in another hosting environment.
- Common reasons for hybrid design include gradual migration, regulatory constraints, specialized hardware, legacy systems, or data that cannot move easily.
- In a hybrid network, traffic often flows between on-premises networks, shared services in AWS, and application VPCs connected through VPN, Direct Connect, or Transit Gateway.
- The challenge is to make the environment feel like one coherent system while still managing latency, security boundaries, address planning, and operational complexity.

Load Balancing in AWS

- A **load balancer** distributes client requests across multiple targets so no single server becomes the only entry point or failure point.
- In AWS, load balancers often sit in public subnets and forward traffic to application instances or containers running in private subnets.
- The main AWS options are **Application Load Balancer** for HTTP and HTTPS traffic, **Network Load Balancer** for high-performance TCP or UDP traffic, and **Gateway Load Balancer** for inserting virtual appliances into the traffic path.
- Load balancing improves availability, supports scaling across multiple Availability Zones, and gives you a stable frontend DNS name even when backend instances change.

Segmentation and Multi-Account Design

- **Segmentation** means dividing infrastructure into smaller trust boundaries so that compromise, misconfiguration, or overload in one area does not automatically spread everywhere.
- In AWS, segmentation can happen at several layers: separate subnets for different tiers, separate VPCs for different systems, and separate AWS accounts for different teams or environments.
- A **multi-account** design is common because AWS accounts provide strong isolation for billing, permissions, service limits, and operational ownership.
- The networking challenge is then to connect only the paths that are truly needed, while keeping defaults restrictive and making cross-account access explicit and auditable.

AWS Accounts, Root, and IAM Users

- An AWS **account** is a top-level administrative boundary with its own billing, resources, quotas, and identity system.
- Every account has a **root user**, which is created with the original signup email address and has full control over the account.
- In normal operations, people should avoid using the root user and instead sign in with **IAM users** or, more commonly in modern setups, IAM roles provided through AWS Identity Center or federated login.
- In a multi-account environment, teams often get separate accounts for development, testing, and production, which reduces blast radius and makes permissions easier to reason about.

Observability and Troubleshooting

- Cloud networking problems are often invisible until you have the right telemetry, so **observability** is a core part of network design rather than an afterthought.
- In AWS, teams commonly use tools such as **VPC Flow Logs**, CloudWatch metrics, load balancer logs, and Route 53 health checks to understand traffic patterns and failures.
- Troubleshooting usually means checking several layers in order: DNS resolution, route tables, security groups, NACLs, load balancer configuration, and application health.
- A recurring challenge is that the network may be technically reachable while the application is still failing, so cloud troubleshooting often requires both network and service-level visibility.

Resilience Across Failure Domains

- AWS networking design should account for **failure domains**, which are parts of the system that can fail independently, such as an instance, a subnet, or an Availability Zone.
- A common resilience strategy is to distribute critical services across multiple AZs so that the failure of one zone does not take down the entire application.
- Network paths also need resilience: for example, using multiple subnets, redundant load balancer targets, and, in hybrid cases, backup connectivity paths.
- Resilience is not free, but the general goal is clear: avoid designs where one route table, one NAT path, one instance, or one AZ becomes the single point of failure.

Cost and Performance Tradeoffs

- Cloud networking decisions involve tradeoffs between **cost**, **latency**, **throughput**, **operational simplicity**, and **fault tolerance**.
- For example, NAT gateways, Transit Gateway attachments, inter-AZ traffic, and Direct Connect can improve connectivity or manageability, but they also add recurring cost.
- Simpler designs are often cheaper and easier to operate, while larger or more resilient designs may require extra network components and more careful address planning.
- Good architecture means choosing the level of complexity that matches the actual requirements instead of assuming that the most elaborate network is always the best one.

Key Takeaways

- AWS cloud networking starts with a small set of core ideas: geographic placement, VPC boundaries, address planning, subnets, routing, and traffic filtering.
- As architectures grow, the main challenge shifts from basic connectivity to safe and scalable interconnection across many services, VPCs, accounts, and environments.
- DNS, load balancing, private connectivity, and observability are not side topics; they are central parts of making cloud systems reachable, secure, and operable.
- Good AWS network design is usually not about using every available feature, but about choosing clear, well-scoped building blocks that match the system’s real needs.